OHJELMISTON UUDISTAMINEN KÄYTÄNNÖN OHJELMISTOPROJEKTISSA

Case: Tremedia Ky

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, syksy 2016

Emmi Lehto

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

VISAMÄKI
Tietojenkäsittely

TIIVISTELMÄ

Tekijä Emmi Lehto **Vuosi** 2016

Työn nimi Ohjelmiston uudistaminen käytännön ohjelmistoprojektissa
Case: Tremedia Ky

TIIVISTELMÄ

Tässä opinnäytetyössä tutkittiin, mitä on ohjelmiston uudistaminen ja uudelleenrakentaminen, sekä miten erilaisten mallien soveltaminen onnistui käytännön ohjelmistoprojektissa. Työn toimeksiantajana toimi Tremedia Ky, joka on tamperelainen digitaalisen median mainostoimisto.

Opinnäytetyössä esiteltiin ohjelmistoprojekti, jossa olemassa oleva ohjelmisto uudistettiin yritykselle. Työ sisältää uudelleenrakentamisen prosessin kuvauksen uuden ohjelmiston esitutkimuksesta betaversion rakentamiseen asti. Työn ensimmäinen osuus käsitteli teoreettisesti ohjelmistotuotannon vaiheita uudistamisen näkökulmasta. Toisessa osuudessa käsiteltiin itse ohjelmistoprojektin prosessin käytännön etenemistä ja käytettyjä tekniikoita.

Opinnäytetyössä oli käytetty ohjelmistotuotannon julkaisuista sekä internetlähteistä saatavilla ollutta tietoa. Lisäksi opinnäytetyössä sovellettiin töissä opittua tietotaitoa ja luotua aineistoa.

Opinnäytetyön lopputuloksena julkaistiin betaversio Tremedia Intra 2016 projektien ja työtuntikirjauksen hallintaohjelmistosta. Ohjelmisto korvaa vanhan Tremedia Intra -ohjelmiston vuodelta 2007. Ohjelmistoa tullaan jatkossa kehittämään yrityksessä, ja se on tarkoitus ottaa käyttöön vuoden 2017 alussa.

Avainsanat ohjelmistotuotanto, verkkosovellus, ohjelmiston uudistaminen, uudelleenrakentaminen, ketterät kehitysmenetelmät

Sivut 59 s. + liitteet 4 s.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Visamäki
Degree Programme in Business Information Technology

Author Emmi Lehto **Year** 2016

Subject of Bachelor's thesis Software re-engineering as part of a practical software project. Case: Tremedia Ky

ABSTRACT

This thesis was about software refactoring and re-engineering. It aimed to find out what techniques and process models would be useful in practical software re-engineering project. The client for this thesis was Tremedia, which is a digital marketing company located in Tampere, Western Finland.

This thesis introduces the software project, where the old software is re-engineered for the client. The thesis includes the process of building a new software. The theory part is about re-engineering and software project techniques in general. The practical part is about the process of implementation of the software.

As the final result of the thesis is the "Intra 2016" project management software's beta release. The final release of the software is going to be done in the early 2017. The software will replace the old software "Intra" from the year 2007.

Keywords re-engineering, software engineering, refactoring, agile methods

Pages 59 p. + appendices 4 p.

SANASTO

Acunetix	Verkkosivustojen tietoturva auditointiin käytettävä haavoittavuuksia löytävä työkalu.
Ajax	Asynchronous JavaScript And XML. Ajax on joukko websovelluskehityksen tekniikoita. Mahdollistaa selainohjelman vaihtaa pieniä määriä dataa palvelimen taustalla niin ettei jokaisen tehdyn toiminnon välissä tule ladata sivustoa uudelleen.
Alfaversio	Ohjelmiston valmistumisasteessa olevan versio. Tässä vaiheessa ohjelmistoa aletaan testata. Ohjelmistoa on mahdollista käyttää, mutta se on toiminnoiltaan pääosin keskeneräinen.
Backend	Ohjelmiston taustan alusta, jonka päälle ohjelmistoa rakennetaan. Ohjelmiston käyttäjä ei näe tätä osaa ohjelmistosta.
Betaversio	Ohjelmiston valmistumisasteessa olevan version julkaisu. Tässä versiossa ohjelmiston alkaa olla toiminnoiltaan ja ominaisuuksiltaan valmis, mutta sisältää todennäköisesti merkittävästi vielä virheitä.
CSRF	Verkkosivuihin kohdistuva hyökkäys, jossa hyökkääjä hyödyntää sivuston luottamusta käyttäjään. Palvelimelle lähetetty pyyntö tulee toiselta käyttäjältä, kuin todelliselta sivuille autentikoidulta käyttäjältä
CSS	Cascading Style Sheets. Tyyliohjeet, joiden avulla muotoillaan erityisesti verkkosivustojen ulkoasua.
Frontend	Ohjelmiston käyttäjälle näkyvä osa, joka käyttää hyödyksi backendin tarjoamia toiminnallisuuksia.
Funktio, metodi	Menetelmä, jolla suoritetaan määrämuotoisesti vähitellen edistyvä toimintoketju.
Inkrementaalinen	Ohjelmisto kehittyy vähitellen, eli koko ajan kasvaen, kohti lopullista muotoaan.
Iteraatio	Menetelmä, joissa samoja työvaiheita toistetaan niin pitkään kunnes toistot halutaan lopettaa.
JavaScript	JavaScript on verkkoselaimessa ajettava dynaaminen ohjelmointikieli.
jQuery	Selaimille tarkoitettu avoimen lähdekoodin JavaScript-kirjasto, jonka etuna on sen helppo ymmärrettävyys.

Luokka	eng. Class. Luokka on olio-ohjelmoinnissa olion tyyppi. Luokka määrittelee ohjeet olion (object) toiminnalle.
Moduuli	Ohjelmistossa toimiva itsenäinen osa, joista on mahdollista koota erilaisia kokonaisuuksia.
MySQL	Web-ohjelmoinnissa käytettävä avoimen lähdekoodin relaatiotietokantaohjelmisto.
Olio-ohjelmointi	Ohjelmoinnin lähestymistapa, joissa ratkaisut jäsennellään olioiden toiminnalla.
Optimointi	Optimoinnin avulla etsitään parasta mahdollista vaihtoehtoa.Esimerkiksi ohjelmistokoodissa etsitään suorituskykyisintä ratkaisua.
Parametri	Parametrit ovat metodeille tai käskyille välitettäviä tietoja, joiden avulla pystytään säätelemään eri tietueita, navigointia ym.
PHP	Hypertext Processor. Ohjelmointikieli jolla voidaan luoda dynaamisia verkkosivustoja.
Phpmyadmin	Selaimessa käytettävä MySQL-relaatiotietokannan hallintatyökalu.
Refaktorointi	Refaktoroinnissa palataan parantamaan huonosti suunniteltua ohjelmistoa ja muuttamaan sitä paremmaksi.
Responsiivisuus	Verkkosivusto joka muuntautuu päätelaitteen näyttökoon mukaisesti.
SQL	Structured Query Language. SQL on standardoitu kyselykieli relaatiotietokannoille.
Tremedia CMS	Tremedia Ky:n omistama sisällönhallintajärjestelmä.
Tremedia Intra	Tremedia Ky:n alkuperäinen hallinto-ohjelmisto vuodelta 2007.
Tremedia Intra 2016	Tremedia Ky:n ohjelmistouudistuksen tuotoksena syntynyt uusi hallinto-ohjelmisto.
Tremedia Verkkokauppa	Tremedia Ky:n omistama verkkokauppajärjestelmä.
XSS	Cross site scripting. Tunnistettu tietoturva-aukko, joka esiintyy usein verkkosovelluksissa. Mahdollistaa koodin syöttämisen ja sen vuoksi mahdollisen tunkeutumisen verkkosivuille.

SISÄLLYS

1 JOHDANTO

Ohjelmistoja tuotetaan yhä enemmän ja ne kehittyvät jatkuvasti. Lisäksi ohjelmistotuotannon trendit muuttuvat, jolloin asetetaan jatkuvasti uusia haasteita ohjelmistojen rakentamiselle. Nykyohjelmistojen rakentamisessa otetaan myös erittäin vahvasti huomioon tuottavuus. Ohjelmistotuotannon ja ohjelmien kehitykseen käytetään paljon pääomaa ja ne ovat olennainen osa yrityksen investointeja. Ohjelmistojen luonnin tärkeäksi osaksi on noussut myös laatu, jota voidaan pitää myös tuottavuuden mittarina.

Vaikka kehitykseen käytetty alkupanostus olisikin tuottava ja laatu olisi saatu ajanmukaisten standardien mukaan korkealle, tekniikat ja käytännöt kehittyvät ohjelmistotuotannon aloilla erittäin nopeasti. Voi siis olla mahdollista, että miljoonaluokan ohjelmiston arvo muuttuu hyvinkin radikaalisti uusien parempien toimintatapojen löytyessä. Myös ylläpitokustannukset voivat nousta erittäin korkeaksi ja käytössä olevat ohjelmistot eivät säästy muutoksilta. Näitä muutoksia on tekemässä yleensä useampi kehittäjä, jolloin laadun varmistaminen ja kustannusten arviointi on hankalaa. Ylläpitoon liittyvää laaduntarkkailua ei tehdä tarpeeksi, vedoten siitä aiheutuviin lisäkustannuksiin ja huonoon organisointiin.

Ohjelmistojen uudistaminen on siis monen ohjelmiston kohdalla välttämätöntä, sillä vanhan ohjelmiston aiheuttama tehottomuus voi estää esimerkiksi yrityksen liiketaloudellisen kehittymisen tai pahimmassa tapauksessa pysäyttää koko yrityksen toiminnan. Ohjelmistotuotannossa sekä ohjelmistojen uudistamiseen, että uudelleenrakentamiseen on kehitetty monia julkisia malleja. Käytännössä kunkin yrityksen tulisi aina ensin itse punnita yrityksen sovellusalueen, olemassa olevan ohjelmiston, sekä liiketaloudellisten tavoitteiden pohjalta, että mitä ohjelmiston uudistamisella voitaisiin saavuttaa. Olemassa olevat mallit toimivatkin hyvänä viitekehyksenä ohjelmiston uudistamiselle, mutta uudistamistarve riippuu paljon ohjelmistosta itsestään.

Ohjelmistojen uudistamisprojektit ovat paljon laajempia kuin esimerkiksi ylläpidon aikana tehdyt muutokset. Uudistamisprojektit muistuttavatkin pikemmin ohjelmistojen alkutuotantoa projektimalliltaan, joten ne sisältävät esimerkiksi kattavan esitutkimuksen sekä vaatimusmäärittelyn. Uudistamisen ymmärtäminen vaatii koko ohjelmistotuotannon mallien ja käytäntöjen tarkastelua laajalla perspektiivillä, sekä sen yhdistämistä yrityksen hankkeisiin ja liiketaloudellisiin tavoitteisiin.

1.1 Opinnäytetyön tavoitteet, tutkimuskysymykset ja rajaukset

Opinnäytetyön tavoitteena on tutkia ohjelmistojen uudistamiseen liittyviä toimintatapoja ja mitä erilasia keinoja uudistamiseen voidaan käyttää. Työ seuraa prosessia, jossa suomalaiselle mediatoimistolle, Tremedia Ky:lle, suunnitellaan ja ohjelmoidaan projektiluontoisesti uusi ohjelmisto. Ohjelmiston uudistaminen on osa Tremedian hanketta, jossa tavoitellaan yrityksen toimintatapojen ja projektien hallinnan parantamista.

Tämän opinnäytetyön lopputuotteena on Tremedia Ky:n uusi hallinto-ohjelmisto projektinimeltään ”Intra 2016”. Ohjelmistossa hallinnoidaan esimerkiksi yrityksen projekteja ja tuntikirjausta. Ohjelmisto toteutetaan web-pohjaisena ja ohjelmoidaan pääosin käyttäen PHP-kieltä.

Opinnäytetyön tavoitteena on havainnollistaa ohjelmistojen uudistamista käytännön pk-yrityksen projektin avulla. Lähtökohtana on uudistamisprojekti, jossa oli mukana opinnäytetyön tekijän lisäksi koko yrityksen henkilökunta, pääosin kuitenkin hallinnollisia toimintoja käyttävä ohjausryhmä. Projektille oli sovittu joustava aikataulu, jonka mukaan lopullisena tavoitteena olisi ohjelmiston laatuarvioitu betaversio vuoden 2016 syksyllä. Ohjelmistoa pyrittiin rakentamaan niin, että siitä oli olemassa jatkuvasti kehitys- ja katselmointiversio. Kehitys tapahtuu paikallisella palvelimella ja työntekijöille julkinen kehitysversio on Tremedian omalla verkkopalvelimella.

Teoriaosuudessa käsitellään yleisesti ohjelmistotuotantoa, siihen liittyviä menettelytapoja sekä avataan ohjelmistojen uudistamiseen liittyviä käytäntöjä ja vaihtoehtoja. Opinnäytetyö sisältää ohjelmistotuotannon eri näkemykset ohjelmistojen uudistamiseen ja tutkii, millä erilaisilla projektimuodoilla ja työmalleilla pystytään tavoittelemaan parhaita tuloksia tämän kaltaisissa projekteissa. Lisäksi perehdytään lyhyesti ohjelmistojen laadun analysointiin osana uudistamisprojektia.

Käytännönosuudessa suunnitellaan ja toteutetaan yritykselle uusi ohjelmisto. Yrityksen vanhan hallinto-ohjelmiston avulla selvitetään tarpeellisia laatutekijöitä. Opinnäytetyössä analysoidaan vanhan, sekä tuotetun järjestelmän laatua samoilla standarteilla.

Opinnäytetyössä pyritään vastamaan seuraavaan tutkimuskysymykseen: Miksi ohjelmistouudistaminen on haastavaa? Tutkimuskysymystä tukevat myös seuraavat tarkentavat kysymykset: Mitä on ohjelmiston uudistaminen? Millainen projektimuoto sopii ohjelmiston uudistamiseen?

Työ on rajattu niin, että sen käytännön osuus sisältää ohjelmiston suunnittelun ja toteutuksen eri vaiheiden analysoinnin. Analysointi alkaa ohjelmistouudistuksen esitutkimuksesta ja päättyy ”Intra 2016”-ohjelmiston betaversion käyttöönoton analysointiin ja siitä saatuihin laatuarvioinnin tuloksiin.

Opinnäytetyön haasteina voidaan nähdä uudistamisprojektin aikataulutus. Opinnäytetyöntekijän tulee ottaa huomioon tämän ohjelmistoprojektin lisäksi useita muita yrityksen ohjelmistoprojekteja joissa hän on ohjelmoijana. Projektityöt mahdollisesti vievät aikaa yrityksen sisäisen ohjelmiston kehitykseltä. Tämä opinnäytetyö on kehittämistyö, eli toiminnallinen työ, jonka tukena on käytetty kirjallisista lähteistä saatua tietoperustaa.

1.2 Toimeksiantajan esittely

Opinnäytetyön toimeksiantaja on Tremedia Ky. Tremedia on Tamperelainen digitaalinen mainostoimisto. Se tuottaa korkeatasoisia ja luotettavia perinteisen ja digitaalisen mainonnan palveluita, jotka perustuvat vahvaan ammattitaitoon ja osaamiseen. Palveluihin ja tuotteisiin kuuluu esimerkiksi verkkosivustot, verkkokaupat ja erilaiset verkkoportaalit. (Tremedia 2016.)

Tremediassa verkkojärjestelmät rakennetaan aina uusimpien tietoturvasuosituksien mukaisesti ja tietoturvaa auditoidaan eri sivustojen kohdalla useasti. Myös responsiivisuus (sivustojen toimivuus eri päätelaitteilla) nähdään yrityksen palveluiden tarjoamisessa tärkeänä laatutekijänä. (Tremedia 2016.)

Tremedia on pieni ja kasvava yritys, jossa toimintamallit yhä kehittyvät. Yrityksen henkilöstö on kasvanut kuluvan vuoden aikana ja projektit ovat laajentuneet mittakaavaltaan suuremmiksi. Toimintamallien kehittäminen asettaa haasteita myös hallinto-ohjelmille, joiden tulisi pysyä uudistuneiden toimintamallien mukana. (Tremedia 2016.)

2 OHJELMISTOJEN UUDISTAMINEN

Ohjelmien tavoitteena on pysyä yritykselle käyttökelpoisena, jolloin ohjelman käyttö tukee yrityksen liiketoimintaa. Ohjelmistojen tuleekin olla alati valmiina vastaamaan yrityksen muuttuneisiin käyttötarpeisiin. Muuttuneet käyttötarpeet voivat olla esimerkiksi kansainvälisten säädöksien tai lakiin liittyvien muutosten mukaisia, tai esimerkiksi liittyä yrityksen hallinnolliseen tai liiketoiminnalliseen muutokseen. (Harsu 2003, 67.)

Muuttuneita käyttötarpeita voidaan muokata ohjelmistoon ohjelmiston ylläpidolla. Ylläpitokustannukset voivat kuitenkin nousta korkeaksi, jolloin ohjelmiston uudistaminen voi olla järkevämpi vaihtoehto ylläpidon helpottamiseksi. Vaikka ohjelmiston ylläpito ja uudistaminen muistuttavat toisiaan, niiden tarkoitus on kuitenkin hyvin erilainen. (Grubb & Armstrong, 2003, 6.)

Uudistustoimenpiteiden tekeminen ohjelmistoihin ei ole yksiselitteisiä. Yritysten tulee usein punnita liiketoiminnalliset tavoitteet, sekä arvioida nykyisen ohjelmiston käyttöarvo. Yrityksille vanhat ohjelmistot ovat usein tärkeitä ja niiden korvaaminen voi olla vaikeaa. Korvaamiseen liittyy yleensä paljon riskejä. Joskus ohjelmiston korvaaminen voi olla yritykselle mahdotonta, jos vastaavaa sovellusta ei löydy markkinoilta tai siihen siirtyminen on olemassa olevan datan siirron vuoksi hankalaa. (Harsu 2003, 65; Grubb & Armstrong 2003, 51.)

Kun ohjelmistoa uudistetaan, on tavoitteena yleensä parantaa ohjelmiston laatua. Tähän voidaan katsoa kuuluvan esimerkiksi ohjelman rakenteellisen laadun parantaminen, kuten ylläpidon tai uudelleenkäytettävyyden parantaminen. Vaikka siis ohjelmisto uudistettaisiin, niin sen ylläpito jatkuu senkin jälkeen. Laatutekijöitä on paljon ja niiden määrittäminen ohjelmiston uudistamisen yhteydessä on tärkeää. (Harsu 2003, 177.)

Kun uudistamista kuvataan prosessina, toistuu siinä samanlaiset vaiheet kuin ohjelmistotuotannon prosesseissakin. Näihin vaiheisiin kuuluu yleensä ainakin määrittely-, suunnittelu-, ja toteutusvaiheet. Muista ohjelmistoprojekteista poiketen ohjelmistojen uudistaminen tehdään aina olemassa olevan ohjelmiston määrityksien pohjalta, jolloin siihen kuuluu esimerkiksi vanhan ohjelmiston arviointia ja analysointia. (Harsu 2003, 165-166.)

Ohjelmistojen uudistamista voidaan pitää laaja-alaisempana kuin ohjelmiston ylläpitoa, eikä näitä tule sekoittaa. Uudistamiseen liittyy paljon enemmän suunnitelmallisuutta, jolloin sitä voidaan projektimalliltaan lähempänä ohjelmistotuotannon projektia. Ohjelmiston ylläpidossa työskennellään vain ohjelmiston olemassa olevien parametrien ja rakentajien parissa. Uudistamisessa puolestaan voidaan luoda myös uusia luokkia tai metodeja ohjelmistoon. (Harsu 2003, 165.; Grubb & Armstrong 2003, 9.)

2.1 Ohjelmistotuotanto yleisesti

Ohjelmistotuotannolla tarkoitetaan ohjelmiston rakentamiseen yleisesti käytettyjä periaatteita, menettelyitä, työkaluja ja tekniikoita. Ohjelmiston

voidaan katsoa kattavan tietokoneohjelman ja siihen liittyvän dokumentaation. (Haikala & Mikkonen 2011, 11.)

Jotta voidaan ymmärtää ohjelmistojen uudistamiseen liittyviä malleja, periaatteita, sekä käytäntöjä, on tärkeää ymmärtää mitä ohjelmistotuotanto on, sekä mistä eri vaiheista se muodostuu. Seuraavien kappaleiden tarkoituksena on pohjustaa lukijalle miten ohjelmistoja luodaan ja mistä ohjelmiston elinkaari koostuu.

Ohjelmistojen tuottaminen tehdään yleensä projektiluontoisena. Projekteissa pyritään toteuttamaan asiakkaalta saatuja vaatimuksia vastaava ohjelmisto. Ohjelmistoprojektin lähtökohtana toimivat aina asiakkaan asettamat vaatimukset. Vaatimuksien kattavuus vaikuttaa siihen miten ohjelmistoa lähdetään kehittämään. (Haikala & Mikkonen 2011, 19-21.)

Ohjelmiston elinkaari kuvaa aikaa, joka kuluu ohjelmiston kehittämisen aloittamisesta sen käytöstä poistamiseen asti. Ohjelmistoja voidaan kehittää erilaisilla malleilla, kuitenkin kaikille näille malleille on yhtenäistä, että ne sisältävät jonkinlaiset ohjelmiston määrittely-, suunnittelu- sekä toteutusvaiheet. (Harsu 2003, 18.)

Nykyohjelmistotuotannossa ohjelmistojen määrittely ei yleensä lähde liikkeelle tyhjästä, vaan ohjelmistot ovat yleensä jossain määrin alustavasti määriteltyjä tai toteutettuja. Ohjelmistot perustuvat yleensä johonkin ohjelmistoalustaan (platform) tai esimerkiksi jo olemassa olevaan aiempaan ohjelmiston versioon. (Haikala & Mikkonen 2011, 23.)

Usein erilaiset ohjelmistot voivat jakaa keskenään myös samoja komponentteja ja kirjastoja, tämä selittää miksi erilaisten ohjelmistojen osien yhdistäminen uudeksi kokonaisuudeksi on mahdollista. Ohjelmistoja pyritään rakentamaan usein myös parametrisoidusti, sekä moduulipohjaisena, jolloin eri osien hyödyntäminen uudelleenkäytössä olisi mahdollista. Lisäksi ohjelmistot sisältävät yhä useammin integraatioita. (Koskimies & Mikkonen 2005, 16.)

Ohjelmiston arkkitehtuurit pysyvät usein miten samanlaisina, jolloin eri sovellusalueen ohjelmistot voivat mahdollisesti jakaa saman perusarkkitehtuurin. Perusarkkitehtuurin jakaminen mahdollistaa myös esimerkiksi nopeamman suunnittelun uudelle ohjelmistolle. (Koskimies & Mikkonen 2005, 16.)

Ohjelmistoprojekteissa voidaan siis todeta uudelleenkäytön merkityksen kasvaneen valtavasti viimeisien vuosien aikana. Kun uusia ohjelmistoja aletaan rakentaa, on niiden suunnittelu aluksi enemmän arkkitehtuuritason suunnittelua. Arkkitehtuuritason käsitteet, sekä toteutusmekanismi määrittelee uuden ohjelmiston suunnittelua pitkälle. (Koskimies & Mikkonen 2005, 16.)

Dokumentointi voidaan nähdä tärkeänä osana ohjelmistosuunnittelua. Myös dokumentaation ylläpito on tärkeää, jotta se pysyy ajan tasalla ja on

hyödyllinen ajatellen ohjelmiston lopputuotosta. Kaikkea ohjelmiston dokumentaatiota ei kuitenkaan kannata ylläpitää, ja sen käytössä tulisikin aina punnita sitä, että mikä on hyödyllistä tietoa ohjelmiston kehityksen kannalta. Dokumentaatio vaatii myös lisätyötä, jolloin kaiken turhan dokumentoiminen kannattaa pyrkiä karsimaan pois. (Haikala & Mikkonen 2011, 194.)

2.1.1 Ohjelmistoprojektin aloitus ja hallinta

Ohjelmistoprojektin organisaatiossa projektin toiminnasta vastaa ohjausryhmä, joka edustaa projektin keskeisimpiä sidosryhmiä. Ohjausryhmän tavoitteena on seurata projektin etenemistä, sekä sen työhön kuuluu mm. vahvistaa ohjelmistoa koskevat tärkeimmät päätökset. Ohjausryhmä on sidoksissa asiakkaaseen, eli ohjelmiston tilaajaan. (Haikala & Mikkonen 2011, 153.)

Toimittajan puolella on määriteltynä projektiryhmä, johon kuuluu projektin jäsenet sekä projektipäällikkö. Lisäksi projektilla voi olla nimetty tukiryhmä, johon voi kuulua esimerkiksi teknisiä asiantuntijoita. (Haikala & Mikkonen 2011, 153-154.)

Asiakasprojekti alkaa yleensä projektin valmistelussa, jossa kuvataan projektin lähtökohtia, tavoitteita, sekä reunaehtoja. Tämän jälkeen projektille valitaan projektipäällikkö ja tekijät, sekä tehdään yksityiskohtaisempi projektisuunnitelma. Kun ohjausryhmä on hyväksynyt projektisuunnitelman, voidaan katsoa projektin alkaneeksi. (Haikala & Mikkonen 2011, 153-154.)

Projektisuunnitteluun kuuluu mm. projektin osittaminen, eli projektin jakaminen hallittaviin osiin. Projektin osittamisen myötä projekti pystytään allokoimaan työntekijöille ja sille pystytään määrittämään aikataulu. Tässä vaiheessa tehdään myös työmäärien arviointia, jonka mukaan voidaan laskea projektille kustannukset. (Haikala & Mikkonen 2011, 157.)

Ketterien ohjelmistoprojektien aikataulutus poikkeaa sillä, että siinä tehdään tarkkoja aika-arviointeja vain lähitulevaisuudessa tapahtuvien ominaisuuksien rakentamisesta. Arvioinnit tarkentuvat mitä lähemmäs ominaisuuden toteutus siirtyy. (Haikala & Mikkonen 2011, 163.)

Ohjelmistoprojektia varten tehdään myös riskienhallintaa. Riskienhallinnassa tunnistetaan riskejä jotka voivat muuttua myöhemmin ongelmiksi. Riskienhallintaan kuuluu mm. riskien tunnistaminen, analysointi, ennaltaehkäisy, sekä suunnitelma riskin toteutumisen varalle. (Haikala & Mikkonen 2011, 164.)

Projektin seurantaa tehdään jatkuvasti projektin aikana. Projektin seurannassa verrataan projektisuunnitelmaan tehtyä aikataulua projektin toteutettuun ja tulevaan aikataulutukseen. Projektin seuranta tapahtuu ohjausryhmän kokouksissa. Projektin seurannalla pyritään havaitsemaan poikkeamat mahdollisimman ajoissa, jolloin niihin pystytään projektin kannalta vielä reagoimaan ja tekemään mahdolliset korjaavat toimenpiteet. (Haikala & Mikkonen 2011, 164.)

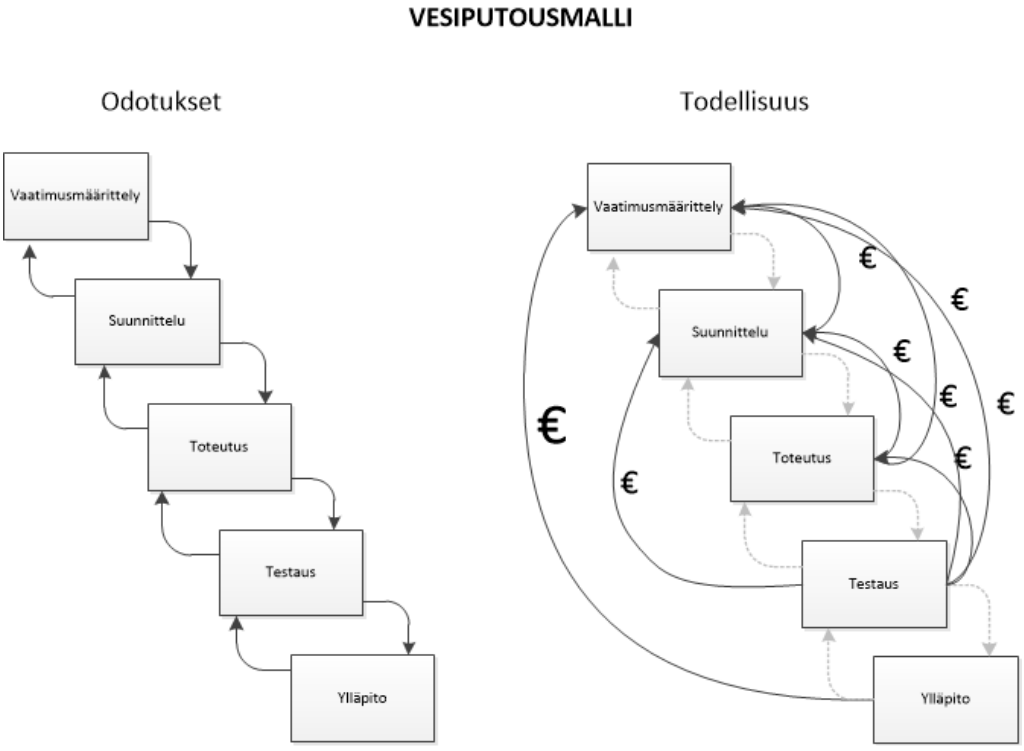
Ketterissä kehitysmenetelmissä projektin seuranta on yksityiskohtaisempaa ja seuranta tehdään projektin edistymiskäyrän (project burndown chart) mukaisesti. Edistymiskäyrä seuraa sekä itse tuotteen valmistumista, mutta myös kyseisen iteraation valmistumista. (Haikala & Mikkonen 2011, 164.)

2.1.2 Projektimallit

Ohjelmistotuotannon projektimalleja on kehitetty paljon vuosien aikana ja osa niistä on myös julkisesti saatavilla. Malleihin liittyen on kasvanut myös erittäin paljon merkittävää liiketoimintaa, joihin liittyy mm. erilaisten sertifikaattien saaminen mallien hallitsemisesta ja järjestettävät koulutukset. (Haikala & Mikkonen 2011, 34.)

Vesiputousmallissa on tarkoituksena iteroida taaksepäin eri vaiheita ja palata näin ollen parantamaan mahdollisen edellisen vaiheen suunnitteluvirheitä. Vesiputousmalli on klassinen ohjelmistotuotannon malli, joka on myös melko vanha. Tätä mallia on usein haukuttu jäykäksi, sillä esimerkiksi ongelmien nouseminen esiin vasta testausvaiheessa voi tarkoittaa ohjelmiston kustannusten kaksinkertaistumista. (Haikala & Mikkonen 2011, 36.)

Kuten kuvassa 1 voidaan nähdä, palaaminen kunkin vaiheen kohdalla taaksepäin voidaan katsoa vaikuttavan ohjelmiston kustannuksiin. Siispä suunnitteluvaiheessa tehty virhe voi aiheuttaa todella suuret kustannukset jos se huomataan vasta testausvaiheessa. Jotta esimerkiksi puutteellinen ominaisuus voidaan korjata, tulee sen käydä läpi uudelleen koko iteraatioiden sarja.



Kuva 1. Vesiputousmallissa myöhemmin huomatut virheet voivat nostaa ohjelmiston kustannukset jopa kaksinkertaisiksi. (Yu 2005)

Ketterät kehitysmenetelmät (agile methods) poikkeavat esimerkiksi vesiputousmallista. Se on yleisnimitys useille ohjelmistokehitysmenetelmille, jotka toki poikkeavat myös toisistaan. Kuitenkin yhtenäistävänä tekijänä voidaan pitää iteratiivisuutta ja inkrementaalisuutta. Ketterät kehitysmenetelmät ovat yleistyneet ohjelmistotuotannon käytössä 2000 – luvulla ja siitä lähtien kasvattaneet suosiotaan. (Lehtonen, ym.)

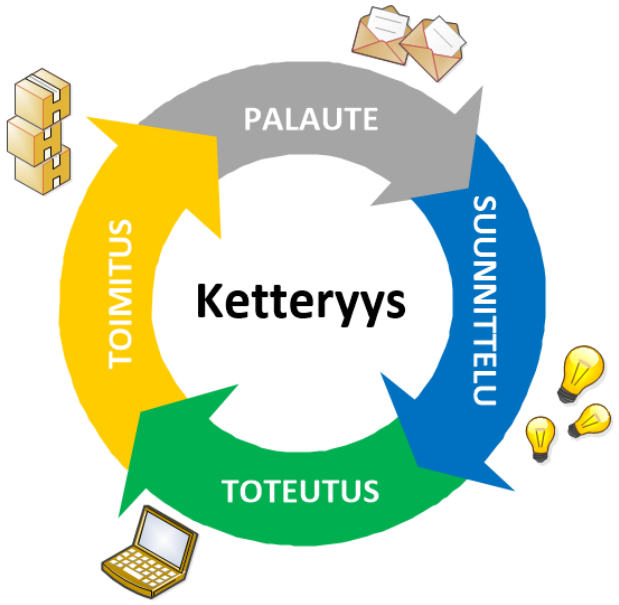
Ketterien kehitysmenetelmien mukaisesti ohjelmistoa kehitetään iteratiivisesti niin, että siihen lisätään ominaisuuksia vähitellen. Ohjelmistoa katselmoidaan myös säännöllisesti ja selkeän kaavan mukaisesti. Ohjelmiston suunnitelma muuttuu ja tarkentuu jatkuvasti, eikä esimerkiksi suunnitelmaa toteuteta aikaisemmin käsitellyn vesiputousmallin mukaisesti kerralla valmiiksi. (Lehtonen, ym.)

Ketterille kehitysmenetelmille voidaan katsoa olevan olemassa erilaisia arvoja, mitä sen käytössä pyritään seuraamaan. Arvot esitelty taulukossa 1.

Taulukko 1. Ketterien kehitysmenetelmien arvot. (Lehtonen, ym.)

Arvo	**Kuvaus**
Yksilö ja kanssakäyminen	Tämän mukaan kehittäjät saavat enemmän vastuuta omasta työstään ja sen suunnittelusta. Kehittäjät ottavat teknisen näkökulman lisäksi kantaa projektin etenemiseen. Projektin toiminnassa pyritään toimivaan kommunikaatioon eri projektin osapuolien kanssa. Työskentelevät yksilöt kommunikoivat keskenään ja tätä tukevat käytetyt työkalut ja menetelmät.
Toimiva ohjelmisto	Ominaisuuksia lisätään inkrementaalisesti. Jokainen ominaisuus ja toiminto on testattu toimivaksi asti. Dokumentaatio tukee työn tekemistä ja ketterien kehitysmenetelmien dokumentaatiossa pyritään keskittymään vain olennaiseen.
Asiakasyhteistyö	Asiakas otetaan suunnitteluun mukaan tiiviimmin, jolloin asiakas pääsee projektin aloituksen jälkeenkin mukaan kehitykseen. Työ esitellään säännöllisesti asiakkaalle, eli sen tulokset katselmoidaan ja asiakas pystyy antamaan palautetta jo kehitysvaiheessa. Molemmat osapuolet pyrkivät jatkuvasti kohti samaa tavoitetta, jolloin molempien osapuolien ymmärrys kehitettävästä tuotteesta lisääntyy.
Vastaaminen muutokseen	Vaatimukset muuttuvat jatkuvasti kehityksen edetessä, jolloin molemmat osapuolet ymmärtävät paremmin ohjelmiston tarpeet. Alkuperäiset suunnitelmat eivät ole yksityiskohtaisia, vaan niitä pyritään jatkuvasti tarkentamaan. Suunnitelmatyö tehdään vain siihen asti mitä

	on tarpeellista, jotta kehitys etenee, mutta ei tehdä turhaa työtä.



Kuva 2. Ketterissä kehitysmenetelmissä pyritään siihen, että myös asiakas osallistuu mm. palautteellaan ohjelman tuotantoprosessiin iteratiivisesti. (Lehtonen, ym.)

Esimerkiksi Scrum on viime vuosien erittäin yleisesti käytetty ketterä kehitysmenetelmä, jonka etuna voidaan nähdä sen perusperiaatteiden yksinkertaisuus. Scrumia ei voida pitää kuitenkaan projektihallintamenetelmänä, vaan se on tapa organisoida projektin sisältämiä iteraatioita sen toteutusvaiheessa. Scrum ei siis yksittäisenä tapana toimi, vaan se vaatii aina muutakin järjesteltyä projektinhallintaa. (Haikala & Mikkonen 2011, 46-47.)

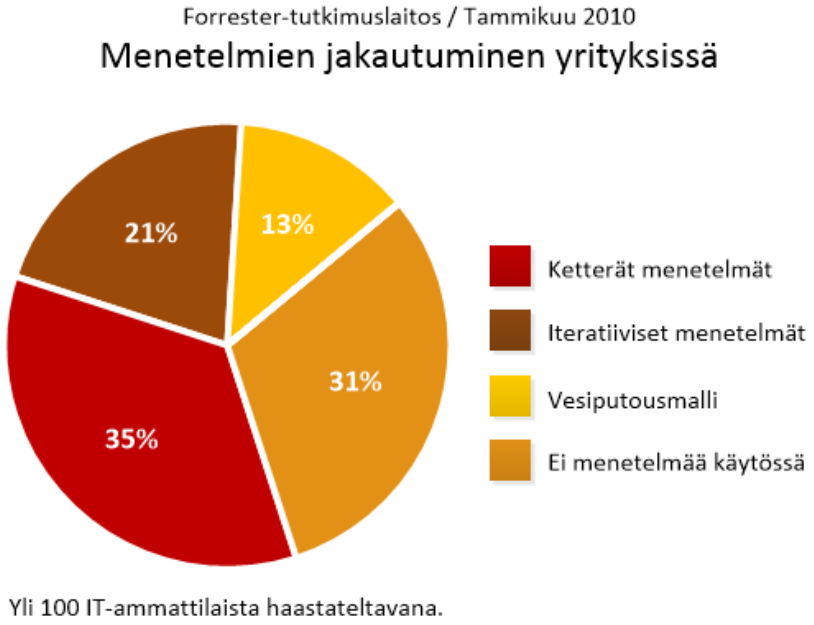
Scrumissa voidaan lyhykäisyydessään kuvailla projektin etenevän pyrähdyksillä (sprint), joiden pituudeksi on määritelty 30 kalenteripäivää. Jokainen pyrähdys aloitetaan aina suunnittelukokouksella, joka kestää koko päivän. Kokoukseen osallistuu koko Scrum tiimi, -mestari sekä tuotteen omistaja. (Haikala & Mikkonen 2011, 48-49.)

Pyrähdyksen eli Sprintin jokaisena päivänä pidetään noin 15 minuuttia kestävä päivän Scrum -kokous (daily scrum). Kokouksen aikana käydään kaikkien projektin jäsenien puolesta läpi, että miten on edetty edellisen kokouksen jälkeen, mitä tehdään seuraavaksi ja onko tullut eteen riskejä tai mahdollisia esteitä jotka hidastavat työn tekemistä (Haikala & Mikkonen 2011, 49.)

Scrum poikkeaa myös muista projektimalleista siten, että myös ohjelmistosuunnittelijat ovat ottaneet sen mielellään käyttöön, verrattain muihin projektimalleihin. Scrumin ottaminen käyttöön vaatii yritysjohdon sitoutumista siihen sekä ajallisesti, että rahallisesti. Tämä johtuu siitä, että sen käyttöä ei pystytä omaksumaan tekemättä muutoksia johtamiskäytäntöihin ja työsken-

telyvälineisiin. Lisäksi se vaatii asianmukaista kouluttautumista henkilöstöltä ja esimerkiksi sertifikaatteihin on suositeltavaa panostaa. (Haikala & Mikkonen 2011, 46-47, 52.)

Mallien käyttäminen vaihtelee yrityksien sovellusalueen, koon, sekä muiden kriteereiden perusteella. Forresterin tutkimuslaitos julkaisi tammikuussa 2010 raportin, joka tutkii ketterien kehitysmenetelmien käyttöönottoa. Siinä oli haastateltu yli 100 IT-ammattilaista sekä heidän käyttämiään menetelmiä yrityksessään. Jakauma voidaan nähdä seuraavassa kuvassa 3. Yllättävää tässä on se, että 31 % haastateltavista ilmoittaa että heillä ei ole yrityksessä käytössä mitään menetelmää. (Haikala & Mikkonen 2011, 52.)



Kuva 3. Menetelmien jakautuminen tutkimuksen mukaan yrityksissä (Haikala & Mikkonen 2011, 52.)

2.1.3 Arkkitehtuurit

Ohjelmistoarkkitehtuuri on tärkeä osa ohjelmistosuunnittelua ja sen olennaisena osana voidaan nähdä siinä pääteltyjen ratkaisuiden perustelut. Ohjelmistoarkkitehtuuri on ikään kuin ohjelmiston perusta, joka määrää tarkemmat rajat yksityiskohtaiseen suunnitteluun sekä toteutukseen, lisäksi se tarjoaa mahdollisuuden tarkastella eri ohjelmiston tai järjestelmän osia eri näkökulmista tai eri abstraktitasoilla. (Koskimies & Mikkonen 2005, 27.)

Näkökulmat eivät myöskään välttämättä kuvaile vain järjestelmää tai ohjelmistoa, vaan arkkitehtuuria voidaan lähestyä liiketoiminnallisesta näkökulmasta, jolloin sitä voidaan kutsua yritysarkkitehtuuriksi (Enterprise architecture, EA). Yritysarkkitehtuurissa määritellään ensin yrityksen liiketoiminnalliset tavoitteet, joihin yrityksen strategian mukaan pyritään pääsemään. Yrityksen ohjelmistojen, sekä sen käsittelemän datan pyrkimyksenä

on auttaa yritystä pääsemään sen liiketoiminnallisiin tavoitteisiin. Yritysarkkitehtuurissa kuvataan miten tieto ja sovellukset tukevat yrityksen toimintaa. (Haikala & Mikkonen 2011, 32.)

Perinteisen ohjelmiston arkkitehtuurin tavoitteena on kuvailla ja ottaa kantaa ohjelmiston keskeisiin ratkaisuihin. Ratkaisuita on esimerkiksi erilaiset prosessit, ohjelmiston osien välillä tapahtuva kommunikointi, toiminnallisuus, tehokkuus, varautuminen erilaisiin tulevaisuuden tarpeisiin ja uudelleenkäytettävyys. Näitä ratkaisuita on todella vaikeaa arvioida etukäteen ja siksi kehitys pysyviin ratkaisuihin tapahtuu vaiheittain. Kehityksen ensimmäiset vaiheet ottavat kantaa ohjelmiston tai järjestelmän perusarkkitehtuuriin ja siitä eteenpäin inkrementaalisesti eri vaiheet tuovat lisää toiminnallisuutta tehtyyn runkoon. (Koskimies & Mikkonen 2005, 19.)

Arkkitehtuuria on myös hyvä arvioida osana uudistamisprojektia. Arkkitehtuurin arviointiin kuuluu esimerkiksi ohjelmiston arviointi pidemmän aikavälin tähtäimellä laajennettavuuden, muunneltavuuden, skaalatutuvuuden arvioinnin perusteella. Suunnittelupäätökset yleensä perustuvat yrityksen ja uuden ohjelmiston liiketoimintatavoitteisiin, jolloin ne toimivat myös arkkitehtuurin arvioinnin perustana. (Koskimies & Mikkonen 2005, 221.)

Arkkitehtuurin tavoitteena ei ole tarkastella esimerkiksi komponenttien sisäistä rakennetta, vaan se pyrkii käsittelemään osien ohjelmiston järjestäytymistä ja niiden välisiä suhteita korkealla abstraktitasolla. Arkkitehtuurin arvioinnissa on tarkoituksena tutkia eri komponenttien ja alijärjestelmien välisiä suhteita kohdennetusti. (Koskimies & Mikkonen 2005, 221.)

Kohdennettu arviointi tarkastelee valittua osaa arkkitehtuurista. Kuitenkaan arkkitehtuurin tarkasteluun ei kuulu, että tarkasteltaisiin yksittäisiä tietorakenteita, algoritmeja tai rajapintojen yksityiskohtia, kuten parametreja. Arkkitehtuurin arvioinnin perusteella voidaan päätellä, että kuinka hyvin ohjelmisto pystyy täyttämään sille asetetut laadulliset vaatimukset. Näitä on esimerkiksi: Suorituskyky, luotettavuus, turvallisuus, muunneltavuus, siirrettävyys ja varioitavuus. Haetut laatuvaatimukset eivät kuitenkaan yleensä ole yksiselitteisiä, joten niitä pitää tarkentaa kunkin ohjelmiston ja järjestelmän kohdalla erikseen. (Koskimies & Mikkonen 2005, 221-223.)

Toiminnallisen vaatimuksien arviointi on mahdollista tehdä myös arkkitehtuurin perusteella. Esimerkiksi voidaan tarkastella, että pystytäänkö kyseisellä arkkitehtuurilla toteuttamaan halutut toiminnot, kuten komponenttien väliset suhteet. Tämän tyyppinen arviointi on usein suoraviivaista ja se on mahdollista osoittaa esimerkiksi sekvenssikaaviolla. (Koskimies & Mikkonen 2005, 223.)

Suunnittelijoiden on helpompaa ymmärtää ohjelmistoa tai järjestelmää paremmin arkkitehtuurin arvioinnin perusteella. Arviointi auttaa myös löytämään ongelmakohtia ohjelmistosta, jolloin niiden muuttaminen esimerkiksi uudistamisprojektissa on mahdollista. Uudistamisprojektin tukena voidaan siis käyttää nykyisen ohjelmiston ainakin osittain automaattisesti luotua arkkitehtuurikuvausta. (Koskimies & Mikkonen 2005, 223.)

2.2 Ohjelmistojen ylläpito

On tärkeää ymmärtää, miten ohjelmiston ylläpito poikkeaa muista ohjelmistolle tehtävistä toimenpiteistä. Seuraavissa kappaleissa käsitellään ylläpitoon liittyviä kriteereitä ja sitä miten ne poikkeavat ohjelmiston muusta uudistamisesta.

Ohjelmiston elinkaareen kuuluu ohjelmistojen ylläpito. Ohjelmiston ylläpidon (software maintenance) voidaan katsoa alkaneeksi heti ohjelmiston käyttöönoton jälkeen. Ohjelmiston ylläpitäminen loppuu kun ohjelma poistetaan käytöstä. Ylläpitoon voidaan nähdä kuuluvan ohjelmiston muuttamistyöt, esimerkiksi uuden ominaisuuden rakentaminen. Ylläpidolla voidaan mm. parantaa ohjelmiston suorituskykyä tai muuttaa ohjelmistoa vastaamaan sille asetettuja muuttuneita vaatimuksia. (Harsu 2003, 18.)

Ylläpidolla ohjelmistoon pyritään tekemään vain pieniä muutoksia. Muutoksien tekeminen ylläpidossa on reaktiivista toimintaa, jossa löytyneeseen puutteeseen pyritään vastaamaan nopeasti. Ylläpidosta poiketen, esimerkiksi uudistamisessa toiminta ja ohjelmiston muokkaaminen on paljon laajempaa ja yleensä myös organisoidumpaa. (Harsu 2003, 30.)

Ohjelmistojen ylläpidolle määritellyt toimenpiteet voidaan jaotella esimerkiksi taulukon 2 mukaisesti.

Taulukko 2. Ylläpitotoimenpiteet ja esimerkit (Harsu 2003, 19.)

Ylläpitotoimenpide	**Kuvaus**
Korjaava ylläpito (corrective maintenance)	Esimerkiksi korjataan virheitä joita käyttäjät löytävät ohjelmistosta sen käyttöönoton jälkeen.
Mukauttava ylläpito (adaptive mainteance)	Esimerkiksi ohjelmisto ei enää vastaa ympäristön asettamiin vaatimuksiin. Se ei enää välttämättä tue käytössä olevaa uutta laitekantaa, käyttöjärjestelmää tai muuta ympäristöä.
Täydellistävä ylläpito (perfective maintenance)	Esimerkiksi käyttäjä huomaa vasta ohjelmistoa käyttäessään tarvitsevansa uuden ominaisuuden ohjelmistoon.
Ehkäisevä ylläpito (preventive maintenance)	Esimerkiksi ohjelmiston koodia eheytetään ja parannetaan jotta tulevat ylläpitotoimet saadaan helpommiksi.

Ohjelmiston ylläpitoon liittyvä tärkein laatutekijä on ylläpidettävyys (maintainability). Ylläpidettävyyteen liittyy mm. se kuinka helppoa on tehdä erilaisia muutoksia kehitettyyn ohjelmistoon, tähän kuitenkin liittyy myös paljon muita laatutekijöitä. (Harsu 2003, 57.)

Ylläpidettävyyden laatua on tärkeää arvioida myös osana ohjelmistojen uudistamista, sillä tällä on suuri vaikutus siihen, että onko uudistaminen kannattavaa. Ylläpidettävyyttä voidaan arvioida laadullisesti tarkastelemalla niitä laatutekijöitä, jotka nostavat sitä. (Harsu 2003, 52-53.)

Ylläpidettävyyttä nostattavina laatutekijöinä voidaan nähdä esimerkiksi taulukon 3 mukaiset kohdat.

Taulukko 3. Ylläpidettävyyttä nostattavia laatutekijöitä (Harsu 2003, 52-53.)

Laatutekijä	**Selitys**
Virheettömyys	Kuinka hyvin ohjelman ominaisuudet täyttävät käyttäjien tarpeet. Noudattaako ohjelma sille annettua määrittelyään.
Käytettävyys	Kuinka helppoa on käyttäjälle oppia ohjelman käyttö. Onko syötteiden, sekä saatujen tuloksien tulkitseminen ohjelmistossa helppoa.
Luotettavuus	Mikä on todennäköisyys sille, että ohjelma toimii odotetusti.
Ymmärrettävyys	Kuinka helppoa on ymmärtää ohjelmiston toimintaa.
Testattavuus	Kuinka helppoa ja tehokasta on testata ohjelmistoa.

Ylläpidettävyyttä voidaan myös tarkastella niin, että tutkitaan tätä heikentäviä syitä. Näitä ovat esimerkiksi ohjelman huono suunnittelu ja toteutus, useiden ohjelmistokielien käyttö samassa ohjelmassa, dokumenttien riittämättömyys ja huono käyttöliittymä. (Harsu 2003, 58.)

Ohjelmiston laadun mittaaminen ja parantaminen aiheuttaa kuitenkin lisäkustannuksia, mutta toisaalta se todennäköisesti vaikuttaa laskevasti tuleviin ylläpitokustannuksiin. (Harsu 2003, 62.)

Vaikka ohjelmiston ylläpidolla voidaan parantaa ohjelmistoa, saattaa ylläpitokustannukset kohota erittäin korkeiksi. Ohjelmiston ylläpito on ohjelmiston elinkaaren pidentämisen kannalta välttämätöntä, mutta ohjelmiston ylläpidolla ei kaikissa muuttuvissa yrityksen tilanteissa pystytä välttämättä muuttamaan ohjelmiston toimintaa halutuksi tai vaikka pystyttäisiinkin, ei se ole kannattavaa korkeiden kustannusten vuoksi.

Ylläpitoon liittyen voidaan pitää vaikeana ohjelmiston muutoksien hallintaa ja seuraamista. Ohjelmiston ylläpitomuutoksien seuraamisen mittarina toimii muutoksen tekemiseen käytetty aika ja vaiva, joka määrittelee sen kuinka paljon joudutaan käyttämään vaivaa koko ohjelmiston ylläpitoon sen varsinaisen käyttöönoton jälkeen. Tutkimuksien mukaan ohjelmiston ylläpitokustannukset voivat olla jopa laajuudeltaan 40 - 70 % koko ohjelmiston elinkaaresta. (Grubb & Armstrong 2003, 7.)

2.3 Vanhat ohjelmistot ja evoluutiolait

Vanhat järjestelmät (Legacy softwares) koostuvat usein ohjelmista, joita on ollut rakentamassa useampi ohjelmoija. Tällöin voidaan katsoa että ohjelmiston koodi on siirtynyt niin sanotusti perintönä ohjelmoijalta toiselle. Näihin ohjelmistoihin on tehty pitkän ajan kuluessa esimerkiksi korjauksia, muutoksia ja muita ylläpidon vaativia toimenpiteitä. Jos kuitenkin ohjelmiston käyttöikä on pitkä, on ohjelmisto osoittanut hyödyllisyytensä. (Harsu 2003, 65.)

Ohjelmien kehittymiselle ja evoluutiolle on olemassa lainalaisuuksia. Lehmanin laeiksi nimetyt huomiot kuvaavat ohjelmistojen evoluutiota ja dynamiikkaa. Seuraavaksi on koostettu muutama esimerkki näistä laeista. (Harsu 2003, 66-67.)

Taulukko 4. Lehmanin lait ohjelmistojen evoluutioista ja dynamiikasta (Harsu 2003, 66-67.)

Laki	**Selitys**
Jatkuva muutos	Järjestelmän tulee muuttua jotta se pysyy käyttökelpoisena. Näin ollen järjestelmään tulevia muutoksia ei siis pystytä välttämään. Tehdyt muutokset voivat mahdollisesti nostaa esiin lisää järjestelmän muutostarpeita.
Lisääntyvä monimutkaisuus	Kehitettävän ohjelmiston rakenne muuttuu jatkuvasti monimutkaisemmaksi, jolloin ohjelmiston rakenteen säilyttämiseen tulee erikseen panostaa. Ehkäisevällä ylläpidolla on mahdollista tehdä korjauksia monimutkaiseen rakenteeseen. Ehkäisevällä ylläpidolla parannetaan ohjelman rakennetta, mutta ei puututa ohjelman toiminnallisiin ratkaisuihin.
Suurten järjestelmien evoluutio	Ohjelman evoluutiota voidaan kutsua itseään sääteleväksi prosessiksi. Tämä tarkoittaa sitä että esimerkiksi järjestelmän koko pysyy muuttumattomana eri versioiden välillä. Suuriin ohjelmistoihin tehdään yleensä harvemmin suurempia muutoksia, sillä niiden tekemiseen liittyy paljon riskejä ja kustannukset ovat korkeita. Suurten muutoksien päätöksiin menee enemmän aikaa ja tämän vuoksi esimerkiksi ohjelmistojen kehitys pysyy suhteellisen tasaisena.
Järjestelmän kehittämisnopeuden muuttumattomuus	Järjestelmän elinkaaren aikana kehittämisvauhti pysyy samana. Kehittämisvauhti on riippumaton käytettävistä resursseista. Eli vaikka ohjelmiston kehittämistiimiin lisättäisiin enemmän ohjelmoija, on tutkittu että sillä ei ole nostattavaa vaikutusta ohjelmiston kehitykseen suorasti.
Samankaltaisuuden säilyttäminen	Eri julkistuksen välillä tehdyt muutokset pysyvät oletuksena suunnilleen yhtä suurina. Jos toimin-

	nallisuuksia lisätään paljon eri ohjelmiston versioiden välillä, vaikuttaa tämä siihen että ohjelmistosta löytyy myös enemmän virheitä. Tästä syystä toiminnallisuuteen vaikuttavat muutokset pyritään pitämään tasaisena eri julkistusten välissä.

2.4 Ohjelmistoihin tehtävät muutostoimenpiteet ja uudistaminen

Muuttuneiden käyttötarpeiden lisäksi on viime vuosikymmenten aikana tapahtunut paljon dynaamisia muutoksia sekä ohjelmistojen suunnittelussa, että rakentamisessa. Ohjelmistojen sisäiset rakenteet ovat muuttuneet ja tämä näkyy esimerkiksi ohjelmistojen välisinä integraatioina. (Fong 2015)

Uudistustarve kunkin ohjelmiston kohdalla voi olla erilainen, mutta yleisesti voidaan nähdä, että uudistettavat ohjelmistot ovat vanhoja. Vanhojen ohjelmistojen rakenteellinen suunnittelu on usein huonoa, eli ohjelmistokoodi ei ole uusimpien mallien tai ratkaisuiden mukainen, sekä ohjelman jatkokehittäminen voi olla huonon ohjelmistoarkkitehtuurin tai alkuperäisen suunnittelun vuoksi vaikeaa. Uudistustarvetta saattaakin olla hankala arvioida. (Harsu 2003, 65.)

Myös jatkuvat muutokset aiheuttavat ohjelmakoodin rakenteen huonontumista. Lisäksi muutoksia on tekemässä useammat ohjelmoijat ja sen seurauksena ohjelmiston kokonaisarkkitehtuurista ei ole välttämättä kehittäjillä enää selkeää kuvaa. Muutoksien tekeminen vaikeutuu entisestään kun ohjelmiston rakennetta ei täysin ymmärretä tai kun siitä ei ole olemassa enää pätevää dokumentaatiota. On myös mahdollista, että ohjelmiston arkkitehtuuriin tai toimintaan liittyy paljon hiljaista tietoa, mikä poistuu yrityksestä vanhan työntekijän mukana jättäen uudet kehittäjät ilman pätevää dokumentaatiota ohjelmistosta. (Harsu 2003, 67.)

Muutostarve ohjelmistojen kohdalla ei tarkoita suoraan sitä, että ohjelmisto olisi huonosti suunniteltu tai toteutettu. Käyttäjien muuttuvat tarpeet kuitenkin tekevät ohjelmistoon liittyvien muutoksien tekemisen välttämättömäksi. Esimerkiksi yrityksessä otetaan käyttöön uusia käytäntöjä, joille ei ole vielä olemassa helpottavia sähköisiä työkaluja tai toimintoja työn tukemiseksi. Muutokset voivat olla myös lakisääteisiä, jolloin niiden tekeminen ohjelmistoon on pakollista. (Harsu 2003, 66.)

Ohjelmiston uudistamiseen voi olla erilaisia ratkaisuita ja yhdeksi vaihtoehdoksi voi toisinaan nousta se, että nykyinen järjestelmä hylätään täysin käytöstä. Tämä tarkoittaa sitä että nykyinen järjestelmä ei ole hyödyllinen yrityksessä liiketoiminnan kannalta. Tämä voi olla tilanne mm. silloin kun yrityksessä on tapahtunut toiminnallisia muutoksia vanhan järjestelmän käyttöönoton jälkeen ja se ei enää palvele yritystä niin kuin aikaisemmin. (Harsu 2003, 73-74.)

Myös järjestelmän ylläpidon jatkaminen saattaa olla vaihtoehto. Tämä voi tulla esiin silloin kun järjestelmää tarvitaan yrityksen käytössä ja se on vakaa, eikä siihen tarvita suuria muutoksia. Tähän liittyen on mahdollista että ylläpidettävyyttä pyritään myös parantamaan tekemällä muutoksia vanhan

ohjelmiston rakenteeseen ja on tiedossa että uusia muutoksia on jatkossakin oletettavasti tulossa järjestelmään. Saattaa olla myös mahdollista että järjestelmä aiotaan korvata kokonaan uudella riippuen sen kustannusten ja hyödyn suhteesta. Joissakin tilanteissa tämä voi olla alustateknisistä syistä myös ainoa vaihtoehto. (Harsu 2003, 74.)

2.4.1 Vanhojen ohjelmistojen analysointi ja laadun evaluointi

Edellisessä luvussa mainitun jatkuvan muutoksen lain vuoksi ohjelmistoja tulee muuttaa niin, että se palvelisi paremmin yrityksen tarpeita. Tämä vaatii sitä, että ohjelmistoa tulee uudistaa tai parantaa. Tätä varten on kehitetty erilaisia käytäntöjä. Uudistamisvaihtoehtoja on erilaisia riippuen siitä, millaisesta ohjelmistosta ja uudistamistarpeesta on kyse. Jotta uudistamisen tarve voidaan selvittää, onkin hyvä analysoida nykyistä ohjelmistoa. (Harsu 2003, 72.)

Ohjelmistojen uudistamisen tarvetta voidaan arvioida tarkastelemalla ohjelmiston laatua ja arvioimalla sen perusteella missä laatu jää vajaaksi. Ohjelmiston laadun määritteleminen voi olla vaikeaa koska se riippuu monesta tekijästä, esimerkiksi ympäristöstä tai sovellusalueesta. Laadun mittaamisen tueksi onkin hyvä tarkastella tekijöitä, joita pidetään hyvälaatuisille ohjelmistoille tyypillisenä. (Tian 2005, 15.)

Ohjelmiston laatu tulee silloin esiin kun ohjelmistoa käytetään sen todelliseen tarpeeseen. Tämä tarkoittaa, että ohjelmiston loppukäyttäjällä on suuri rooli ohjelmiston laadun määrittelyssä. Loppukäyttäjä arvioi että onko tuote helppokäyttöinen ja vastaako se niihin tarpeisiin johon hän ohjelmistoa tarvitsee. Myös muut ohjelmiston sidosryhmät pystyvät määrittelemään ohjelmiston laatua. Esimerkiksi palvelun kehittäjät ja ylläpitäjät analysoivat onko ohjelmiston rakenne selkeä ja sen vuoksi helposti kehitettävissä tai päivitettävissä. (Tuovinen 2016)

Laadun mittaamiselle on olemassa erilaisia malleja ja standardeja, joiden mukaan ohjelmiston laatua pystytään mittamaan. Esimerkiksi olemassa oleva SQuaRE (Software Quality Requirements and Evaluation), joka on yleisnimi ISO/IEC 25000 -tuoteperheelle. SQuaRE malli sisältää ohjelmiston laatumallin, jossa on joukko laadun mittareita ohjelmistolle ja järjestelmälle. Malleja on useita ja niitä tulisi käyttää kohdennetusti, eli esimerkiksi isommilla ja pienemmillä yrityksillä laadun mittaukseen käytetty malli ei voi olla useimmiten sama. (Kan 2002; Tuovinen 2016)

Ohjelmistojen laatumittarien tarkoituksena on keskittyä ohjelmistotuotteen laadun mittaamiseen. Laatumittareita voidaan jakaa useampaan luokkaan, seuraavaksi yhden näkemyksen mukaan määritellyt kategoriat: Kehitysprosessin laadun mittarit, lopputuotteen laadun mittarit ja ylläpidon laadun mittarit. (Kan 2002)

Laatua alentavat ohjelmistossa olevat virheet. Yleensä on kyse inhimillisistä virheistä ohjelmistoprosessin aikana. Esimerkiksi määrittelyvaiheessa

voi olla mahdollista, että tehdään suunnitteluvirhe ohjelmistoon. Virheestä johtuen ohjelmistoon voi syntyä vika tai ongelmatila, mikä puolestaan voi aiheuttaa ohjelmistossa häiriötilan ja aiheuttaa ongelmia käyttäjille. (Kipponen 2005)

Seuraavaksi on kuvailtu yleistä laatupiirteiden jakautumista eri tietojärjestelmissä ja ohjelmistoissa. Sitä voidaan käyttää suoraan ohjelmiston laatuarviointiin.

Taulukko 5. Ohjelmiston laatumallin esimerkki. (Tuovinen 2016)

Taso	Kuvaus
Toiminnallinen sopivuus (Functional suitability)	Ohjelmiston toiminnot kattavat niille suunnitellut tehtävät ja vastaavat käyttäjän tarpeisiin. Esimerkiksi ohjelmisto toimii oikein, eli tuottaa täsmälliset ja oikeat halutut tulokset.
Tehokkuus (Performance efficiency)	Suorituskykyä arvioidaan tehtävien suorittamiseen ja verrataan käytössä olevien resurssien määrään. Esimerkiksi vaste-ajat ovat ohjelmistolle määriteltyjen nopeusvaatimuksien mukaiset ja ohjelmiston kapasiteetti on riittävä.
Yhteensopivuus (Compatibility)	Ohjelmisto tai sen osa pystyy kommunikoimaan toisen ohjelmiston osan tai laitteen kanssa. Esimerkiksi ohjelmisto jakaa integroidun järjestelmän kanssa tehokkaasti yhteistä tietoa.
Käytettävyys (Usability)	Ohjelmiston käyttäjät pystyvät suoriutumaan ominaisuuksien käytössä mielekkäästi, tehokkaasti ja tuottavasti. Esimerkiksi ohjelmiston käyttäjä löytää tarvittavat toiminnot nopeasti ohjelmistosta.
Luotettavuus (Reliability)	Ohjelmisto selviytyy sille määrätyistä toiminnoista määritellyissä olo-suhteissa. Esimerkiksi ohjelmisto on toipumisvalmis, eli se pystyy virheen jälkeen palauttamaan muutoksen alaisena olleen tietosisällön.
Turvallisuus (Security)	Ohjelmiston tietosisältö on suojattu sille vaaditulla tavalla. Esimerkiksi ohjelmiston kautta ei ole ulkopuolisten mahdollista saada arkaluonteisia henkilötietoja.
Ylläpidettävyys (Maintainability)	Ohjelmistoa pystytään ylläpitämään tehokkaasti tai tuottavasti. Esimerkiksi ohjelmiston moduuleissa voidaan hyödyntää uudelleenkäyttöä helposti
Siirrettävyys (Portability)	Ohjelmisto on mahdollista siirtää toiselle laitteistolle tai erillaiseen käyttöympäristöön tehokkaasti ja tuottavasti.

2.4.2 Miksi ohjelmistoja tulee muuttaa tai uudistaa?

Muutos- tai uudistamistarpeen toteaminen kunkin ohjelmiston kohdalla voi olla haasteellista niiden monimuotoisuuden vuoksi. Ohjelmistojen muutostarpeet tulevat yleensä paremmin esiin kun sitä analysoidaan ja arvioidaan. Ohjelmiston arviointiin voidaan käyttää edellisessä luvussa esiteltyjä laatumallin osia, joiden perusteella on konkreettisempaa tehdä uudistamiseen liittyviä päätöksiä. Lisäksi on tärkeää laatumallin lisäksi ymmärtää mikä on ohjelmiston asema yrityksessä ja sen sovellusalueella.

Lähestytään ohjelmistojen uudistamistarvetta tarkastelemalla tämän opinnäytetyön toimeksiantajan omistamaa Tremedia Intra -järjestelmää, joka on palvellut yritystä vuodesta 2007.

Järjestelmää voi kuvata aiemmin kirjoitetun luvun 2.2.1 mukaisesti perintöä sisältäväksi ohjelmaksi, johon useampi ohjelmoija on jättänyt monen vuoden aikana oman kädenjälkensä. Ohjelmisto on täynnä erilaisia toimintoja ja ohjelmistopohjassa on käytetty paljon erilaisia menetelmiä vanhoista tekniikoista aivan uusiinkin web-ohjelmointitapoihin (Esimerkkinä AJAX). Ohjelmistosta ei ole olemassa dokumentaatiota ja ohjelmiston koodi on pitkälti kommentoimatonta.

Ennen kuin uusi ohjelmisto (Tremedia Intra 2016) tullaan ottamaan käyttöön, vanha ohjelmisto (Temedia Intra) on elinkaarellaan ylläpito - vaiheessa. Kun ohjelmisto poistetaan käytöstä, sen elinkaari päättyy. Vanhan ohjelmiston ylläpitoon kuluu aikaa, sillä ohjelmistoon pitää tehdä muutoksia, jotta se palvelisi paremmin yrityksen muuttunutta tarvetilaa. Näiden muutoksien ja korjauksien tekemiseen kuluu kuitenkin todella paljon aikaa, sillä pohjan muokkaaminen ja nykyisten ominaisuuksien ymmärtäminen on vaikeaa. Ohjelmiston joidenkin uusien ja tarpeellisten ominaisuuksien rakentamiseen menisi liikaa aikaa, tai niiden tekeminen olisi mahdotonta pohjakoodin vuoksi. Tämän vuoksi osa ylläpidon aikana tulleista muutoksista toimintoihin on jätetty kokonaan toteuttamatta tai hylätty niiden suunnitteluvaiheessa.

Ohjelmiston käyttöliittymä ja toiminnot ovat hyviä ja tarpeeksi yksinkertaisia, mutta pohjakoodi estää ohjelmiston kehittämistä ja uusien toimintojen tekemisen. Miten ohjelmistoja tulisi kehittää jotta se palvelisi muuttuvaa ja alati kasvavaa yritystä? Jotta voidaan selvittää, minkälainen tarve yrityksellä on nykyisen ohjelmistonsa käytössä, tulee tätä vanhaa ohjelmistoa ja yrityksen tarpeita analysoida.

Voidaan päätellä, että Tremedia Intra – ohjelmistossa uudistamisen tarve näkyy ohjelmiston ylläpidon sekä sen kehityksen ongelmallisuuksissa, joihin ei ole olemassa suoraa ratkaisua. Tämä tarkoittaa, että yrityksen sisäisen ohjelmiston käyttöön kuluu paljon aikaa, josta yritys ei saa yhtään tuottoa. Ylläpidon laatua ei tarkkailla ja sen laatua pidetään jopa toisarvoisena, sillä sen laadun nostaminen vaatisi uudistamistoimenpiteitä. Ohjelmistoa ei voida suoraan päivittää tai muuttaa niin, että se palvelisi paremmin, sillä sen koodipohja on huonosti rakennettu ja ohjelmiston kehittäminen on todella vaikeaa.

Ohjelmisto ei myöskään toiminnoiltaan vastaa enää yhtä hyvin käyttäjien tarpeisiin ja käyttäjillä kuluu tarvittavaa enemmän aikaa ohjelmiston käyttämiseen. Lisäksi ohjelmisto on hidas käyttää (mm. hakutoimintojen hitaus, suorituskyky on matala) ja siitä löytyy myös jonkin verran korjaamattomia virheitä (bugit häiritsevät käyttäjää ja vievät työltä tehokkuutta).

Voidaan todeta, että esimerkkitapauksemme pohjalta ohjelmistoja tulee muuttaa, koska niiden ympäristö vaatii muutoksen olevan alati jatkuvaa.

2.4.3 Ohjelmistojen uudistaminen

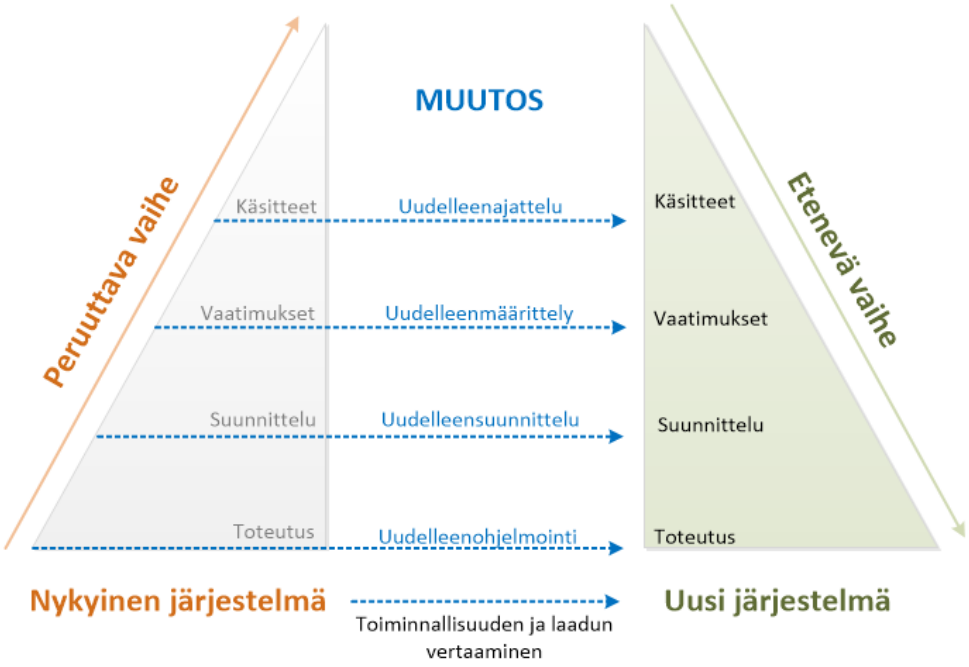
Uudistamisessa (re-engineering) tarkoittaa ohjelmiston uudelleenmuokkaamista, sekä sen uudelleensuunnittelua. Käytännössä uudistaminen voi olla kuitenkin myös kokonaisvaltaista ohjelmiston uudistamista. (Harsu 2003, 26.)

Uudistamisen on esitelty useita määritelmiä, yksi näistä on vuonna 1993 R.S. Arnoldin IEEE:ssä esittelemä määritelmä jonka mukaisesti uudistamista voi kuvailla toimintana, jonka ansiosta ohjelmiston laatu paranee ja se tulee paremmin ymmärrettäväksi. Lisääntynyt laatu näkyy esimerkiksi ohjelmiston parempana ylläpidettävyytenä ja sen uudelleenkäytettävyyden tason lisääntymisenä. (Harsu 2003, 24-25.)

Joidenkin muiden uudistamisen määrittelyiden mukaan, uudistamisessa keskitytään vain ohjelmiston lähdekoodin parantamiseen. Saattaa olla että, määrittelyn mukaan uudistamisessa parannetaan vain ohjelmiston sisäisiä mekanismeja puuttumatta ohjelman toiminnallisuuteen. Tämä käytännössä voi tarkoittaa, että ohjelmiston uudistamisessa ei pyritä puuttumaan sen toiminnollisiin tai määrittelyssä tapahtuneisiin ongelmallisuuksiin tai suunnitteluvirheisiin. (Harsu 2003, 24-25.)

Uudistamiselle on olemassa monia määrittelyitä, joiden merkitys käytännön uudistamistyössä voi olla hyvinkin erilainen. Määrittelyitä sekä malleja kannattaa siis tutkia, mutta kuitenkin uudistamiseen tulisi valita sopivat käytännöt aina uudistettavan ohjelmiston mukaisesti.

Kuvassa 4 esitellään uudistamisen abstraktitasojen muutosta ja esitellään yleisen mallin avulla ohjelmistojen uudistamista. Uudistamiseen abstraktitaso muuttuu, kun edetään uudistamisen eri vaiheisiin. Korkeimmilla tasoilla tarkastellaan sovellussuuntautuneita käsitteitä ja puolestaan matalimmilla tasoilla toteutuksen yksityiskohtia. Abstraktitasolla siirryttäessä alaspäin, vastaa se ohjelmistotuotannon eri vaiheiden läpikäyntiä. (Harsu 2003, 25-27.)



Kuva 4. Yleinen malli ohjelmistojen uudistamiseen (Harsu 2003, 26.)

Kuvassa 4 nähdään kuinka uudistamiselle esitetään kolme erilaista vaihetta. Peruuttavassa vaiheessa voidaan käyttää apuna esimerkiksi takaisinmallintamista nykyisen järjestelmän analysointiin. Tästä edetään muutosvaiheeseen, jossa ohjelmistoon tehdään muutoksia, niin että pysytään jatkuvasti samalla ominaisuuksia esittävällä abstraktitasolla. (Harsu 2003, 26-27.)

2.4.4 Refaktorointi

Ohjelmistotuotannossa yleisenä ajatuksena on, että ohjelmistot ensin suunnitellaan ja sen jälkeen ne ohjelmoidaan, suunnitteluun ei enää palata ja usein vain todetaan, että huonosti suunniteltu ohjelmisto on myös huonosti ohjelmoitu. (Fowler;Beck;& Brant 2002, 9.)

Vaikka ohjelmiston suunnitteluun käytettäisiin paljon aikaa ja se ohjelmoitaisiin sen mukaan, ei ohjelmisto silti säästy ylläpidon aikana tulleista muutoksista ja parannuksista. Muutokset muokkaavat alkuperäistä ohjelmistoarkkitehtuuria ja on mahdollista, että ohjelmiston rakenne kärsii epäselvistä ja kiireellä tehdyistä muutoksista, joita vielä kaiken lisäksi useampi ohjelmoija saattaa tehdä ohjelmistoon. Refaktoroinnissa (Refactoring) palataan parantamaan huonosti suunniteltua ohjelmistoa ja muuttamaan sitä paremmaksi. Refaktorointia voidaan myös, olio-ohjelmiin kohdistuneena, kutsua olio-ohjelmien uudistamisen luokkakokoelman uudelleenmuokkaamiseksi. (Harsu 2003, 22.; Fowler;Beck;& Brant 2002, 46-47.)

Refaktorointia voidaan pitää pitkälti ohjelmistokoodin siivoamisena. Ohjelmistokoodia siivotaan sen vuoksi että ohjelmiston rakenne saadaan paremmin ymmärrettäväksi ja muokattavaksi. Refaktoroinnin tärkeimmät tavoitteet ovat ohjelman luokkien rakenteen selvä parantaminen, sekä luokkien

rajapintojen järkeistäminen (Harsu 2003, 22; Fowler;Beck;& Brant 2002, 47.)

Ohjelmistoon refaktoroinnin seurauksena itse ohjelmistokoodiin voidaan tehdä paljon muutoksia, jotka kuitenkaan eivät näy loppukäyttäjille ollenkaan. Refaktroinnissa ei ole tavoitteena optimoida tai parantaa ohjelmiston toimintaa vaan ohjelmiston toiminnot pysyvät samana ja loppukäyttäjä ei huomaa ohjelmistossa eroa refaktroinnissa ja sen jälkeen. (Fowler;Beck;& Brant 2002, 46-47.)

2.4.5 Takaisinmallinnus

Takaisinmallinnusta (reverse engineering) voidaan pitää tutkimisena ja analysointina, jossa pyritään ymmärtämään ohjelmiston eri komponentteja, sekä niiden välisiä suhteita. Ohjelmaa pyritään esittämään eri muodossa tai korkeammalla abstraktitasolla. Toisin kuin muissa ohjelmistouudistamisen menetelmissä, ohjelmistoon ei tehdä muutoksia. (Harsu 2003, 22.)

Takaisinmallinnuksessa on mahdollista erotella ainakin kaksi erillistä osa-aluetta ja nämä ovat uudelleendokumentointi (redocumentation) ja suunnitteluratkaisun jäljittäminen (design recovery). (Harsu 2003, 22.)

Uudelleendokumentoinnissa tarkastellaan vanhoja dokumentteja ohjelmistosta, muokataan näitä vanhoja dokumentteja ja luodaan myös uusia dokumentteja jotka vastaavat ohjelmiston nykyistä muotoa. Suunnitteluratkaisun jäljittämisessä on kyse tietojen keräämisestä ohjelmistosta. Tähän kuuluu ohjelman toiminnan tai sovellusalueen tietojen keräämistä, kuten esimerkiksi ohjelman lähdekoodin tarkastelua, ohjelman käyttäjäkokemuksien analysointia. Suunnitteluratkaisun jäljittämisellä tunnustetaan ohjelmiston korkeimman tason abstraktioita. (Harsu 2003, 23-24.)

Takaisinmallintaminen auttaa myös todentamaan ohjelmiston laatua ja sen avulla pystytään ymmärtämään ohjelmiston suunnitteluratkaisuita paremmin. (Harsu 2003, 24.)

2.4.6 Uudelleenkäyttö

Uudelleenkäyttö (software reuse) on tärkeässä roolissa sekä ylläpidossa, että uudistamistoimenpiteissä. Uudelleenkäyttöä voidaankin pitää ohjelmistotekniikan yhtenä tärkeänä päämääränä. Käytännössä tämä tarkoittaa, että kaikkea jo rakennettua ohjelmistotekniikkaa tulisi olla mahdollista soveltaa myös uudempiin ohjelmistoprojekteihin. Uudelleenkäyttö olisi hyvä ottaa huomioon myös aina ohjelmistosuunnittelussa. (Itkonen 2003)

Ohjelmistoista on olemassa yleinen käsitys, jonka mukaan jopa 60 - 80 % siitä koostuu aikaisemmin rakennetusta ohjelmistosta. Uudelleenkäyttöä voidaankin pitää erittäin hyvänä keinona lisätä ohjelmistojen rakentamisen tuottavuutta. Uudistamisen apuna voidaan oikeastaan käyttää mitä tahansa

ohjelmistotuotannon vaihetta tai aiemmin kehitysvaiheessa syntynyttä tuotosta. (Haikala & Mikkonen 2011, 190.)

Uudelleenkäytössä hyödynnetään alkuperäiseen ohjelmistoon tuotettua osaa, uuden ohjelmiston (tai saman ohjelmiston) uutta osaa varten. Käytännössä uudelleen käytettävä osa voi olla esimerkiksi samankaltaisen projektin määrittelyssä tuotettu dokumentti, jota nyt hyödynnetään vastaavanlaiseen uuteen projektiin. Useimmiten uudelleenkäyttö on kuitenkin komponenttitason uudelleenkäyttöä tai ohjelmistokoodin uudelleenkäyttöä (code reuse). Yleisin tapa on ohjelman eri komponenttien kierrättämien. Komponentteja kuitenkin tulee yleensä muokata niin, että ne sopivat uuteen käyttötarkoitukseen sopivaksi. (Haikala & Mikkonen 2011, 190.)

Yleisiä kierrätettäviä komponentteja ovat yleensä käyttöliittymän rakentamiseen tarkoitetut kirjastot, laskentaan tarkoitetut kirjastot tai tietorakennekirjastot. (Haikala & Mikkonen 2011, 191.)

Toteutuksen pohjalle otetaan usein komponentin vanha versio ja sitä muokataan niin, että saadaan aikaiseksi uuteen käyttötarkoitukseen sopiva komponentti. Yleensä on mahdotonta muokata kaikkia komponentteja ja variaatioita niin että ne saadaan toimiviksi, siksi komponentteja voidaan myös parametrisoida tai käyttää ehdollista kääntämistä jolloin voidaan hyödyntää samaa komponenttia poikkeavilla käyttötarkoituksilla. Komponenttitason geneerisyyttä on helpompaa toteuttaa hyödyntämällä perintää, sekä erilaisia malleja (templates) avuksi kehityksessä. (Haikala & Mikkonen 2011, 190-192.)

Komponenttitason uudelleenkäytön ongelmina voidaan nähdä siihen liittyvän työtaakan lisääntyminen. Työtaakka voi esimerkiksi näkyä siinä että uusia ohjelmakirjastoja joudutaan luomaan ja ylläpitämään, tai ohjelmistoon komponenttien etsimiseen kuluu aikaa. Lisäksi ohjelmoijat voivat olla haluttomia käyttämään muiden kehittäjien rakentamia komponentteja, sillä haluavat käyttää vain omia tuotoksiaan niiden paremman ymmärrettävyyden vuoksi tai sen vuoksi, että ohjelmoija tekee itse nopeammin uuden komponentin kuin tutustuu toisen kehittäjän aikaansaannokseen. Saattaa olla että luotu komponentti on rakennettu liian yleiskäyttöiseksi, jolloin se vie ohjelmistolta paljon muistia toimiakseen ja siksi on toiminnallisuudeltaan hidas ja jäykkä. (Haikala & Mikkonen 2011, 192.)

2.4.7 Tietojen uudistaminen

Tietojen (mm. tietokantojen) uudistaminen (data re-engineering) on olennainen osa koko ohjelmiston tai järjestelmän uudistamista. Tietojen uudistamiseen kuuluu mm. eri sovellukset, tietokannat ja rajapinnat. (Fong 2015; Harsu 2003, 239.)

Tietoihin kohdistuva uudistaminen on erityisen tärkeää koko ohjelmiston kannalta, sillä esimerkiksi tietokannassa esiintyvät ongelmat heijastavat suoraan koko ohjelmiston muuhun toimintaan. Tietokannan uudistamisen tarve tulee esiin yleensä silloin kuin esimerkiksi useat ylläpitotoimenpiteet ovat aiheuttaneet erilaisia ongelmia. (Harsu 2003, 239-240.)

Tietoihin liittyviä ongelmia on esimerkiksi määrittelyyn tai arvoihin liittyvät ongelma. Määrittelyyn liittyviä ongelmia on esimerkiksi tietoalkioiden epäyhteneväinen nimeäminen, tietueiden kenttäpituuksien poikkeavuus, tai esimerkiksi tietueiden epäyhteneväisyys. Arvoihin liittyviä ongelmia on esimerkiksi epäyhteneväiset oletusarvot, kuten esimerkiksi poikkeamat MySQL tietokannassa arvolla nolla, eli NULL. Joissakin tietueissa tämän merkintä tapa on 0 kun taas toisaalla se on NULL. Toinen esimerkki arvoihin liittyvistä ongelmista on isojen ja pienten kirjainten erottaminen, johon saattaa olla käytetty hyvinkin epäyhtenäisiä käytäntöjä, tämä voi aiheuttaa ohjelmistossa virheitä joita on vaikea havaita. Kehittäjät itse saattaa olettaa että muut kehittäjät käyttävät samoja käytäntöjä. (Harsu 2003, 240-242.)

Tietokannan takaisinmallintamisessa (database reverse engineering) voidaan ottaa vanhan ohjelmiston tietokanta avuksi uuden kehittämiseen. Takaisinmallinnuksen apuna on useita valmiita työkaluja, joita voidaan käyttää. Esimerkiksi Microsoft Visio tarjoaa takaisinmallintamiseen toiminnon jossa tietokanta voidaan ladata ohjelmaan SQL-muodossa ja se muodostaa siitä muokattavan visuaalisen näkymän. (Microsoft, ei pvm)

Myös tietokantojen uudistamiseen on kehitelty erilaisia malleja, esimerkiksi relaatiotietokantojen muuttaminen oliotietokannoiksi. Oliotietokannoissa puhutaan että tietokannan taulu vastaa oliontietokannan luokkaa ja sen sisällä olevat käsitteen attribuutit vastaavat silloin luokan attribuutteja. Tällöin on mahdollista esim. erotella attribuutteja yliluokalle tai aliluokalle. (Harsu 2003, 248.)

Ohjelmistojen uudistamiseen kuuluu myös jossain määrin tietueiden, eli datan siirto. Tämänlainen tarve voi syntyä silloin kun ohjelmisto vaihdetaan kokonaan toiseen tai kun ohjelmistosta tuotetaan uusi tietokantarakenteeltaan radikaalisti poikkeava versio. Kun uusi ohjelmisto tai sen versio halutaan käyttöönottaa, halutaan tällöin yleensä tuoda vanhat tietueet käyttöön uuteen ohjelmistoon, jolloin tärkeät tiedot eivät jää vanhaan ohjelmistoon vaan ovat suoraan käytössä.

2.5 Ohjelmistojen uudistamiseen liittyviä esimerkkejä

Kuten aikaisemmista luvuista voidaan päätellä, eivät uudistamishankkeet ole koskaan yksiselitteisiä ja uudistamiseen on olemassa paljon erilaisia käytäntöjä riippuen ohjelmiston sovellusalueesta, laajuudesta, rakenteesta ja muista ominaisuuksista. Ohjelmisto voi olla osana suurta järjestelmää, jossa on lisäksi laitteistoita ja alihankkijoita yhteydessä. Ohjelmistoihin liittyy paljon ulkopuolisia tekijöitä liittyen lakiin tai esimerkiksi tietoturvaan.

Laatu-arviointi on vaikeaa ja siinä tulee ottaa huomioon yrityksen toiminta jo liiketoiminnallisista tavoitteista ja hankkeista lähtien. Arvioinnin perusteella pyritään tekemään päätös ohjelmiston uudistamisesta, muokkaamisesta, ylläpidon jatkamisesta tai ohjelmiston hylkäämisestä. Uudistamishankkeiden tai ylipäätään IT-projekteihin liittyvät ongelmat eivät ole harvinaisia, yleensä nämä liittyvät projektin aikana esiin tulleisiin ongelmiin, joista johtuen projektien aikataulut venyvät ja kustannusarviot ylittyvät.

Ohjelmistojen, varsinkaan hallintojärjestelmien, epäonnistuneet hankkeet eivät päädy yleensä julkisuuteen. Kuitenkin jotkut julkishallinnon epäonnistuneet IT-hankkeet voivat päättyä median otsikoihin. Ohjelmistotuotannon-projekteissa käytetään samoja menetelmiä, prosesseja, johtamiskäytäntöjä, sopimusmalleja ja teknologiaa, joten tästä voidaan päätellä että samat virheet toistuvat myös yksityisen sektorin hankkeissa. (Järvenpää & Kankare 2013, 75-76.)

Epäonnistuneista hankkeista on mahdollista ottaa opikseen. Seuraavissa kappaleissa käsitellään suomessa tapahtuneita järjestelmä- ja ohjelmistouudistamiseen liittyviä hankkeita, jotka ovat olleet myös julkisuudessa.

Case: VR GROUP, Lippujärjestelmän uudistaminen, vuosina 2008–2015. VR-lippujärjestelmän uudistaminen oli vuonna 2008 käynnistetty hanke. Tavoitteena hankkeelle oli uudistaa koko junayhtiö VR Groupin myyntijärjestelmä sekä päätelaitteisto. Ohjelmistosuunnittelussa otettiin käyttöön myös uusi lippujen hinnoittelun dynaamisuus. Järjestelmän rakennukseen osallistuivat ohjelmistoyhtiöt Accenture, Tieto ja Enfo. (Järvenpää & Kankare 2013, 109-110.; Saarinen 2015)

Käyttöönottotavoite ohjelmistolle oli vuoden 2011 maaliskuussa. Käyttöön ottoa kuitenkin siirrettiin kahteen kertaan hankkeen myöhästymisen takia, ensin vuoden 2011 kesäkuulle ja lopulta saman vuoden syyskuulle. Kyseessä oli 15 miljoonan euron hanke ja on oletettavaa että alkuperäiset kustannusarviot ylittyivät paljon. Lisäksi käyttöönoton jälkeen syntyi niin paljon ongelmia, jotka aiheuttivat palvelun katkonaisuutta, virhetilanteita ja muita häiriöitä. Käyttäjät olivat tyytymättömiä palveluun ja media haukkui palvelun laatua. (Järvenpää & Kankare 2013, 109-110.)

Tämän lisäksi vuonna 2015 uutisoitiin VR:n tekevän 2,7 miljoonan euron uudistamisen tietojärjestelmäänsä. Uudistus ostettiin samalta yhtiöltä, Accenturelta, jolta VR osti aikaisemman uudistamisen järjestelmään. Käytännössä yhtiöllä ei ollut muuta vaihtoehtoa, sillä alkuperäisen uudistamisen tehnyt Accenture omistaa järjestelmän ohjelmistokoodin ytimen, jolloin muut yritykset eivät pystysi kehittämään järjestelmää. (Saarinen 2015)

Koska järjestelmä on yhden yrityksen kehitettävissä, on VR Groupin mahdotonta kilpailuttaa tulevien uudistuksien kustannuksia ja on sidonnainen yhteen toimittajaa. Jos ongelmia syntyy toimittajan kanssa ja he haluavat vaihtaa sitä, joutuu VR Group käytännössä ostamaan kokonaan uuden järjestelmän. (Saarinen 2015)

VR:n uusi järjestelmä tuli kaikesta päätellen kalliiksi yritykselle ja tämän lisäksi yrityksen imago kärsi varmasti paljon.

Case: Sampo Pankki uudistaa tietojärjestelmänsä, vuonna 2008. Toisena esimerkkinä kalliista järjestelmäuudistamisesta voidaan pitää Sampo pankin vuonna 2008 tekemää järjestelmäuudistusta. Järjestelmäuudistamisen tärkeimpänä tehtävänä oli päivittää verkkopankin tietoturvaratkaisuita. Tarkoituksena oli korvata Sampo pankin tietojärjestelmä emoyhtiö Danske

Bankin järjestelmillä. Tietojärjestelmäuudistuksen hinnaksi on arvioitu 200 miljoonaa euroa. Käyttöön oton aikana vuoden 2008 keväällä ilmeni niin paljon ongelmia, joiden aikana pankin asiakkaat kärsivät eniten verkkopalveluiden häiriöistä. (Laitinen 2008)

Järjestelmän virheiden lisäksi, myös esimerkiksi maksukorteissa oli paljon häiriöitä johtuen alihankkia IBM:n ongelmista. Ongelmat kestivät useamman kuukauden verran ja kilpailijoiden arvioiden mukaan Sampo Pankki mahdollisesti menetti jopa 40 000 asiakastaan järjestelmäongelmien vuoksi. Esimerkiksi OP arvioi saaneensa muutaman kuukauden aikana ainakin 18 000 uutta asiakasta ja heidän mukanaan siirtyneen mm. noin 200 miljoonan euron arvosta lainoja. Tietojärjestelmän uudistaminen ja siitä koituneet virhetilanteet ja ylläpidon kustannukset tulivat siis kaikesta päätellen todella kalliiksi Sampo pankille ja lisäksi pankin maine kärsi erityisesti median myllytyksen seurauksena. (Lappalainen 2008; Iivonen 2008; Vanhala 2012)

Ohjelmistohankkeiden ongelmaksi on muodostunut niin projektimuotojen toimimattomuus, kuin myös kustannustehottomuus. Lisäksi kustannustehottomuutta lisää arviointivirheet, joiden seurauksena ohjelmistojen julkaisutavoitteet ovat myöhästyneet, budjetti on ylittynyt monesti miljoonilla euroilla ja on tehty kriittisiä virheitä jotka ovat myöhemmin aiheuttaneet yritykselle korkeat korjaus- ja ylläpitokustannukset. Kuten aikaisemmin todettu, ylläpitokustannusten suurentuminen vaikuttaa suuresti siihen, miten ohjelmiston tuottavuus kärsii.

Projektien asiantuntija-arviointeja ja konsultteja on kritisoitu, sillä budjetit ovat ylittyneet tuntuvilla summilla. Lisäksi on kritisoitu projektimuotoja, joiden uskotaan olleen suuri vaikuttaja siihen, että projektien budjetit, eivätkä aikataulut ole pitäneet. Aikataulun ylitys voidaan melkein suoraan nähdä verrannaisena ohjelmiston kustannuksien kasvamiseen. Myös järjestelmän käyttöön ottoon liittyvät ongelmat ja resurssien aliarviointi tuntuu jatkuvasti yllättävän toimittajat. Esimerkiksi kummassakin edellä mainitussa projektissa olisi ollut ehkä mahdollista jakaa ohjelmiston käyttöönottoa pidemmälle aikavälille, jolloin kehittäjillä olisi ollut aikaa korjata mahdollisia virheitä ja varata tarpeeksi resursseja muutostoimenpiteitä varten.

Onnistuminenkin on mahdollista ja järjestelmiä varmasti pystytään tuottamaan edullisemmin ja tarkemmin ketterillä kehitysmenetelmillä. Valitettavaa on, etteivät onnistumiset välttämättä päädy median otsikoihin. Ohjelmiston rakentaminen alkaa jo toimittajan valinnan perusteella, jolloin toimittajien palveluihin erilaisiin vaihtoehtoihin tutustuminen on tärkeää.

3 TREMEDIA INTRA 2016 -HANKKEEN LÄHTÖKOHDAT JA TAVOITTEET

Tremedia Intra -ohjelmiston uudistamisen tarve oli noussut esiin useamman vuoden ajan yrityksessä. Ohjelmistoon oli haluttu useampia hallintoa helpottavia toimintoja, mutta niiden toteuttaminen on ollut hankalaa, koska ohjelmiston alkuperäinen pohja on rakennettu vanhalla ohjelmointitavalla, eikä siinä ole käytetty esimerkiksi olio-ohjelmointia lähes ollenkaan.

Tremedia Intra -ohjelmiston käyttäminen on todella hidasta ja tietokantarakenne on huonosti rakennettu. Datan ja käyttäjien määrä on kasvanut ohjelmistossa, joka on aiheuttanut ohjelmiston hidastumista ja sen suorituskyvyn heikkenemistä. Ohjelmistossa käsitellään suuria määriä indeksoimattomia tietojoukkoja, joissa on paljon dataa. Arkistointi-ominaisuuksia tai muita pitkän aikavälin datan eheyttämistoimintoja ei ole nykyiseen ohjelmistoon tehty ollenkaan. Nykyisestä ohjelmistosta löytyykin sen käyttöönoton jälkeen kaikki kirjatut projektit, työtunnit ja muu data viimeiseltä yhdeksältä vuodelta. Käytännössä tämä tarkoittaa sitä että esimerkiksi hakutoiminto hakee jokaisella sanalla 9 vuoden tietojoukkojen joukosta kaikki vastineet.

Ohjelmisto on käyttöönotettu vuonna 2007 ja sitä alun perin kehittänyt ohjelmoija on suunnitellut myös ohjelmiston ulkoasun ja käyttöliittymän. Koska alunperäinen ohjelmoija ei ole enää yrityksessä töissä, on suuri osa ohjelmiston tuntemusta (hiljainen tieto) poistunut myös yritykseltä.

Käyttöönoton jälkeen ohjelmistolle on tehty vain ylläpitoon liittyviä muutoksia ja korjauksia. Lisäksi on rakennettu joitakin tarvittavia ominaisuuksia, kuten työtuntien siirtoon liittyviä toimintoja. Ylläpitoon liittyen ei ole tehty dokumentointia eikä laaduntarkkailua, joten myös muuttuneet toiminnot alkuperäisversiosta ovat epäselviä nykyisille kehittäjille. Ohjelmistoon ei ole tehty refaktorointia, eikä muitakaan kontrolloituja eheyttämistoimenpiteitä.

Tremedia Intra -ohjelmistoa on rakennettu aina alkuperäisen version backend-pohjan päälle, jonka vuoksi viimeisimpinä vuosinakin tehdyt osat ohjelmistossa eivät ole hyvällä tekniikalla tai logiikalla toteutettuja. Tekijöitä, sekä tapoja tehdä on ollut useampia vuosien saatossa.

Nämä edellä mainitut asiat ovat vaikeuttaneet ohjelmistoon tehtävien muutoksien tekemistä ja vanhojen toimintojen muuttamista. Lisäksi huono tekniikka ja logiikkatoteutus ovat aiheuttaneet sen, että osa ohjelmiston osista tai komponenteista toimii erittäin hitaasti.

Tremedia Intra -ohjelmisto ei enää palvele toiminnoiltaan yritystä samalla lailla kuin yhdeksän vuotta sitten. Yritys on laajentunut viimeisien vuosien aikana ja työntekijöiden määrä on kasvanut. Yrityksen hallinnointiin liittyvät työt ovat vaikeutuneet koska hallinnoitavat projektit ovat laajentuneet vuosien aikana, lisäksi hallinnoijia on tullut lisää. Projektinhallinta on myös kokenut muutoksia, jotka vaativat tuekseen toimivampaa ohjelmistoa.

Uuden ohjelmiston tavoitteena on tukea tehokasta työtä joka edesauttaa sekä työntekijöiden kuin myös hallinnon suoriutumista tehtävistään. Ohjelmiston on tarkoituksena toimina osana yrityksen liiketoiminnallisten tavoitteiden saavuttamista.

3.1 Uudistusvaihtoehtojen arviointi

Uudistusvaihtoehtoja lähestyttiin tarkastelemalla ohjelmiston tietokantaa, josta selvisi ensimmäiset ohjelmistoon liittyvät ongelmat ja uudelleenajattelua vaativat relaatiot, eli yhteydet. Tavoitteeksi muodostuikin ensin aloittaa ainakin osittain tietokannan takaisinmallinnuksella, jonka pohjalta olisi helppo suunnitella uutta ohjelmistoa toimivammaksi.

Refaktorointi jouduttiin sulkemaan pois uudistamisvaihtoehdoista, sillä sen tehokas käyttö olisi vaatinut olio ja luokkapohjaisen järjestelmän, jota vanha ohjelmisto ei pääosin ole. Refaktorointi ei myöskään olisi riittänyt yksittäisenä tapana parantamaan ohjelmiston toimivuutta. Tähän vaikuttaa mm. se että tietokantaan tarvittiin suuria muutoksia, jotta ohjelmiston arkkitehtuurista olisi järkevämpi. Vanhan ohjelmiston toimintalogiikkaa arvioidessa eteen nousi jatkuvasti se, että koko ohjelmiston perusajatusta ja arkkitehtuuria olisi parempi muuttaa toimivammaksi.

Ohjelmistokoodin vaihe vaiheelta uudistaminen olisi ollut liian työlästä. Esteeksi nähtiin kuitenkin liian sekainen koodipohja, tietokantarakenne ja backendin logiikan puutteellisuudet. Uudistamiseen tarvittava työmäärä alkoi selvitä vanhan ohjelmiston analysoinnin perusteella, jolloin todettiin että vanhan ohjelmistopohjan muokkaaminen ei olisi järkevää.

Koska yrityksellä on käytössään paljon erilaisia moduulipohjaisia olio- ja luokkarakenteella toteutettuja verkkoportaaleja, oli loogisin vaihtoehto uudelleen käyttää muiden portaalien koodeja tehokkaasti. Tämä todettiin hyväksi vaihtoehdoksi, sillä vaikka ohjelmiston kehittäminen aloitettaisiin niin sanotusti tyhjästä, olisi silti olemassa vanhan ja tietokannan ohjelmiston takaisinmallinnuksesta saadut tulokset joiden avulla pystyttäisiin tekemään uutta ohjelmistoa uudelle pohjalle varsin tehokkaasti.

Myös vanhan ohjelmiston koodeja pystyttäisiin analysoimaan kehityksen aikana, jolloin kehityskelpoisen koodin integroiminen uuteen ohjelmistoon olisi mahdollista ja osa ominaisuuksista voitaisiin siis uudelleen käyttää muutamien muutoksien jälkeen.

Uuden ohjelmiston kehittäminen täysin uudella pohjalla, uudelleen käyttäen vanhan ohjelmiston suunnittelu- sekä koodimateriaaleja, valittiin uudistamistavaksi tässä projektissa. Päätös tehtiin ohjausryhmän, opinnäytetyön tekijän sekä teknisen asiantuntijan kesken esitutkimuksen jälkeen.

3.2 Projektimallin valitseminen

Projektille valittiin toteutustavaksi ketterät kehitysmenetelmät, joista sovellettaisiin yritykselle sopivalla tavalla iteratiivisuutta, sekä inkrementaalisuutta. Se oli perusteltua sillä, että uutta ohjelmistoa ei olisi mitenkään pystytty ennalta määrittelemään valmiiksi kerralla ja toteuttamaan sen pohjalta. Ketterät kehitysmenetelmät painottavat myös sitä, ettei ominaisuuksia tai toimintoja suunniteltaisi liian tarkasti alustavan aikataulun yhteydessä, vaan kunkin iteraation lähestyessä pystyttäisiin kutakin toimintoa tarkentamaan inkrementaalisesti.

Uudistamiseen liittyi paljon epäselviä kohtia, esimerkiksi siitä että miten vanhan ohjelmiston ohjelmakoodia olisi mahdollista uudelleen käyttää uudessa ohjelmistossa. Ohjelmiston kehittäminen iteratiivisesti mahdollisti myös ohjausryhmälle paremman aseman projektin seuraamiseen ja lisäksi kehittäjän oli helpompi allokoida ja osittaa projektia muiden asiakastöiden ohessa. Ominaisuuksien karkea määrittely, ja iteraation lähestyessä tai sen aikana tehty tarkempi määrittely, mahdollisti myös projektin nopean aloituksen, joka oli tämän projektin kannalta tärkeää.

3.3 Motivaatio hankkeen toteutukseen

Motiiveina tässä työssä toimi se, että työ on ohjelmistotyö, jossa opinnäytetyön tekijä pystyi kehittämään sekä taitojaan ohjelmistojen suunnittelijana ja kehittäjänä. Työ voitiin nähdä mielenkiintoisena, sen vuoksi että se tutkii mitä kaikkea ohjelmistojen uudistamistyössä tulee ottaa huomioon.

Opinnäytetyön tekijällä oli mahdollisuus tutustua myös projektinhallintaan ja oppia uusia asioita projektien organisoinnista käytännön tasolla. Tärkeään ja kokeilulliseen rooliin nousi myös ketterien kehitysmenetelmien soveltaminen. Opinnäytetyön tekeminen mahdollisti myös kokeilla sopisiko vapaamuotoisiin iteraatioihin liittyvä toimintatapa yritykselle myös jatkossa.

Opinnäytetyön tuote, Tremedia Intra 2016, ohjelmisto oli tärkeä uudistus jota tarvittiin yrityksen käyttöön. Sen arveltiin tehostavan projektinhallintaa huomattavasti ja todellinen hyöty nähtäisiin seuraavan vuoden lopussa jolloin pystyttäisiin tekemään jo tarkempia data-analyysejä kerätyistä tiedoista. Tarkemmat analyysit mahdollistaisivat yritykselle paremman tuottavuuden kun uusia asiakasprojekteja suunniteltaisiin paremmin.

3.4 Kehityksen pääkohdat ja suurimmat ongelmat

Uuden ohjelmiston suunnittelun alussa oli tärkeää koota ne asiat mitä uudessa ohjelmistossa haluttiin parantaa aiempaan nähden. Kehityksen pääkohdat koskivat sekä teknistä toteutusta kuin myös ohjelmiston käytettävyyttä. Kehityksen pääkohtia voidaan pitää ohjelmistoprojektin alustavina laatuvaatimuksina joiden täyttämiseen pyritään ohjelmiston ensimmäisestä kehitysversiosta lähtien.

Tietokannan analysoinnin ja takaisinmallinnuksen yhteydessä esiin nousi kysymyksiä siitä, miten tietokanta voitaisiin toteuttaa mahdollisimman toimivaksi. Tavoitteeksi muodostui datan eli tietueiden parempi järjestely jo ruohojuuritasolla, eli tietokannan tauluissa. Paremmalla järjestelyllä voitaisiin saavuttaa parempi rakenne joka vaikuttaa ohjelmistokoodin optimointiin, esimerkiksi tietokantakyselyiden rakentamiseen. Uutta tietokantaa suunniteltaessa pyrittiin ottamaan myös vanhan datan siirron mahdollisuus huomioon. Jolloin kaikki vanha data olisi mahdollista siirtää uuteen järjestelmään tarvittaessa mahdollisimman helposti.

Tietokantarakenteessa haluttiin myös ottaa huomioon indeksointi, jota aiemman ohjelmiston tietokannassa ei hyödynnetty. Indeksoinnin etuna on se, että tietokanta pystyy toimimaan nopeammin. Kantaan suunniteltiin myös relaatioyhteyksiä, jolloin pää- sekä viiteavainten rooli korostuisi uudessa rakenteessa. Relaatioyhteyksien etuna voidaan nähdä mm. kannan eheänä pysyminen, rakentamisen aikaisten virheiden syntymisen estäminen. Lisäksi relaatioyhteyksillä rakennetut tietokannat ovat yleisesti helpompia ymmärtää.

Ohjelmarakenteen uudistamisessa tärkeänä osana nähtiin yleisesti olio-ohjelmoinnin, sekä luokkien, että funktioiden käyttäminen ohjelmiston rakentamisessa. Laatutekijänä ohjelmarakenteen uudistamisessa voidaan pitää myös ohjelmointikoodin kehityskelpoisuutta. Kehityskelpoisuuteen liittyen nähtiin tärkeäksi se, että tulevassa järjestelmässä ohjelmistokoodi olisi paremmin kommentoitua ja helposti muokattavissa. Lisäksi haluttiin, että ohjelmistokoodi on helposti uudelleenkäytettävissä ja moduuleita pystyttäisiin käyttämään useampaan tarkoitukseen parametrien tai muutamien muutoksien avulla.

Ohjelmarakenteessa haluttiin myös panostaa ohjelmiston kehittämistä niin, että se olisi tietoturvallisempi kuin aikaisempi versio. Lisäksi haluttiin painottaa integrointimahdollisuuksia, tarkoittaen tällä, että olemassa olevaa ja hyväksi todettua ohjelmistokoodia (esim. moduulia tai metodia) hyödynnettäisiin kehityksessä mahdollisimman tehokkaasti ja älykkäästi. Tämä tarkoittaa käytännössä sitä että ohjelmistokoodia otettaisiin tarpeiden mukaan myös täysin erillisen sovellusalueen ohjelmistoista, kuten esimerkiksi verkkokaupan hallintaportaalista.

3.5 Projektin rajaus

Projekti sisälsi web-ohjelmiston kehityksen suunnittelusta betaversion toteutukseen. Projekti ei sisältänyt ohjelmiston tietoturvatestausta, loppukäyttäjätestausta, datan siirtämistä tai käyttöönottoa.

Projektissa tehtiin alustavaa suunnittelua myös vanhan ohjelmiston datan, eli tietokannan tietojen siirtoa uuteen ohjelmistoon, sillä ainakin asiakastietojen pohja olisi hyvä siirtää järjestelmään. Suunnitelmaan kuului myös karkeasti tiedot siitä, että miten käyttöönotto voitaisiin toteuttaa niin, ettei se häiritsisi yrityksen toimintaa radikaalisti.

Projektin etenemisen yhteydessä vastaan tulleet jatkokehitysideat laitettiin erilliselle kehityslistalle. Kehityslistalla olevien lisätoiminnallisuuksien toteuttaminen rajautui tämän projektin ulkopuolelle.

Ohjelmiston betaversion hyväksyi Tremedian johtoryhmä, eli projektin ohjausryhmä. Hyväksynnän jälkeen ohjelmistoprojekti todettiin tämän määritellyn rajauksen mukaisesti valmiiksi. Projektin vastuuhenkilönä toimi opinnäytetyön tekijä Emmi Lehto.

3.6 Projektin dokumentointi

Projektin dokumentointi tapahtui suunniteltujen iteraatioiden eli kehityssyklien mukaisesti. Kahden viikon iteraatioiden (n. 20 h työskentelyä projektin parissa) välein koostetusta raportista selviää miten ohjelmiston rakentaminen on edennyt. Raportissa seurattiin onko projekti pysynyt aikataulussa, sekä tehtiin mm. laadunvalvontaan ja riskien tunnistamista.

Projektin etenemistä seurasi myös ohjausryhmä, jolle ohjelmiston etenemisestä raportoitiin viikon tai kahden välein. Tarkemmin projektin etenemistä käsiteltiin myös noin kolmen viikon välein pidetyissä kehityspalavereissa joissa paikalla oli pääasiallisesti ohjausryhmän jäsenet ja kehittäjä.

4 TREMEDIA INTRA 2016 -OHJELMISTON SUUNNITTELU JA TOTEUTUS

Ohjelmiston kehitys aloitettiin suunnittelemalla aikataulu projektille hyväksytyn esitutkimuksen jälkeen. Aikataulu jaettiin kahden viikon mittaisiin iteraatioihin. Aikataulun suunnittelun yhteydessä huomioitiin projektin rajaus, jolloin tiedettiin jo alustavasti mitkä ominaisuudet ja toiminnot jäävät tämän projektiaikataulun ulkopuolelle.

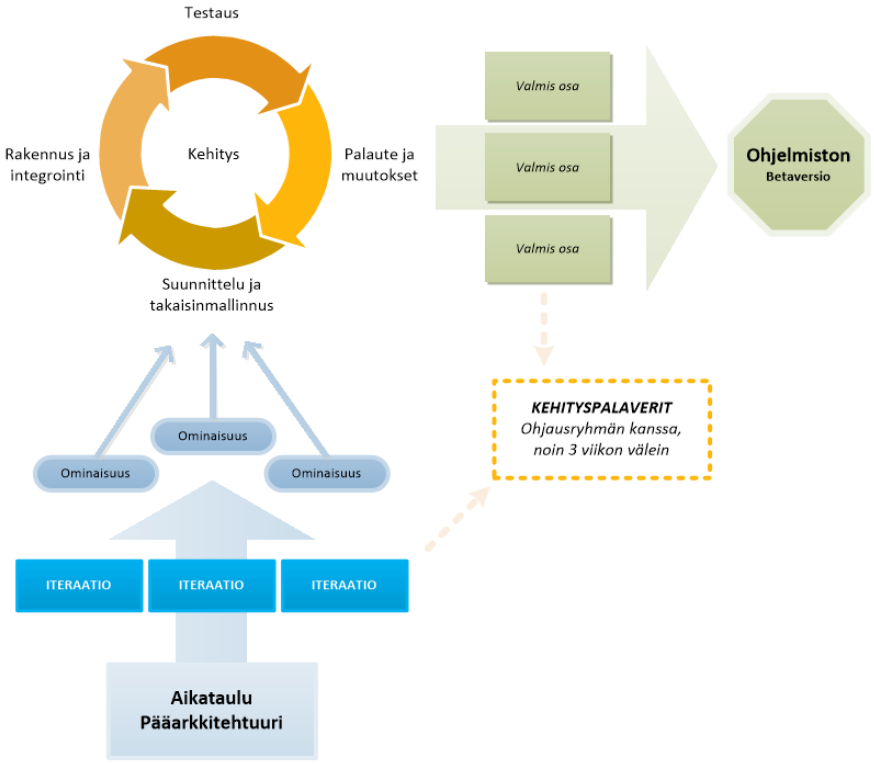
Projektin aikataulu suunniteltiin ketterien kehitysmenetelmien mukaisesti. Se suunniteltiin aina pääpiirteittäin tietyllä viikkomäärällä ja aina lähestyttäessä kunkin iteraation kohdalla pystyttiin tarkentamaan tehtäviä asioita. Asioita lähinnä listattiin erilliselle tehtävälistalle (task backlog), jossa niitä sitten suunniteltiin ja toteutettiin inkrementaalisesti.

Aikataulu muuttui projektin aikana kahteen kertaan radikaalisti: ensimmäisen kerran keväällä 2016 ja toisen kerran kesän 2016 aikana. Aikataulu oli alustavasti suunniteltu liian tiukasti ja yrityksen muut asiakasprojektit veivät kehittäjän aikaa ohjelmistoprojektilta. Aikataulun löysääminen kahteen kertaan mahdollisti sen, että opinnäytetyön tekijä ei ollut pelkästään sitoutunut yhden projektin ohjelmointiin ja viikkotunneista jäi paremmin aikaa sisäisen ohjelmiston rakentamiseen.

Kun päätös ohjelmistohankkeen toteuttamisesta oli tehty, muutettiin samalla myös nykyisiä projektinhallintokäytäntöjä yrityksessä. Käytäntöjen muuttaminen tehtiin lähinnä sen vuoksi, että yrityksen työntekijöiden olisi nopeampaa mukautua uudenlaiseen projektien hallinnointitapaan. Käytännössä tämä tarkoitti sitä että käytössä olevassa Intra 2016 ohjelmistossa jätettiin käyttämättä muutamia erilaisia toimintoja, kuten yksittäisten projektiin kuulumattomien tehtävien käyttäminen merkintätapana ja projektille suoraan tuntien merkitseminen.

Aikataulutuksen yhteydessä tehtiin myös projektin osittaminen kunkin iteraation kohdalla. Projektimuodoksi valttiin ketterien kehitystapojen soveltaminen, jolloin uutta ohjelmistoa suunniteltaisiin, ohjelmoitaisiin ja testattaisiin jatkuvasti kunkin iteraation kohdalla. Ominaisuuksia tarkennettaisiin aina ennen niiden aloittamista ja kustakin iteraatiosta pyrittäisiin pitämään kehityspalaveri ennen sen aloittamista. Kehityspalavereissa käytäisiin myös läpi ohjelmistoon valmistuneita osia, koska niiden perusteella saattoi tulla esiin muutoksia liittyen tuleviin iteraatioihin ja niiden sisältämiin ominaisuuksiin.

Palautteen saaminen kunkin iteraation kohdalla oli tärkeää, jotta ominaisuudet saatiin vietyä siihen suuntaan kuin haluttiin. Suunnittelussa oli tavoitteena käyttää rautalankamalleja hyväksi, jolloin ominaisuuksien kuvaileminen ohjausryhmän kokouksissa olisi helpompaa ja vaatimusmäärittelyä olisi helpompi tehdä kunkin iteraation kohdalla.



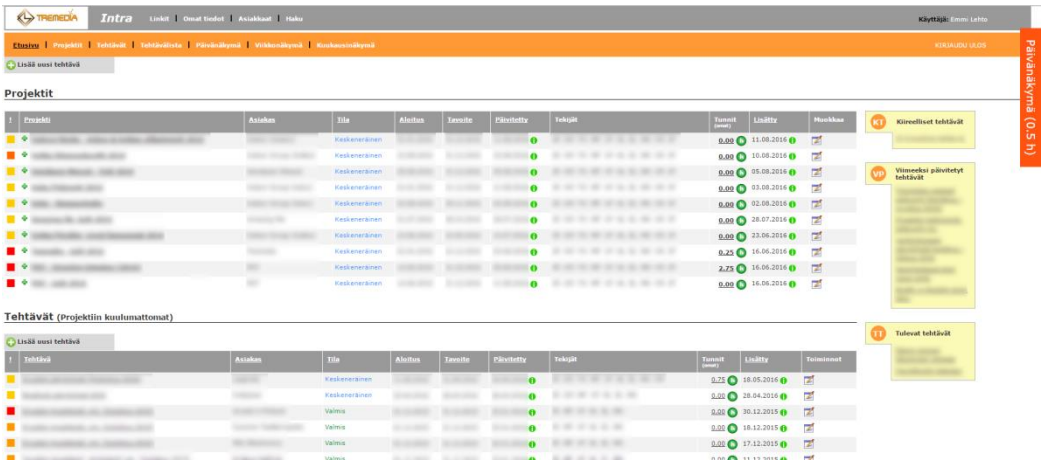
Kuva 5. Ohjelmiston kehitystä kuvaava kaavio. Ohjelmiston kehitys ei noudata mitään yleistä mallia, mutta mukailee ketterien kehitysmenetelmien menettelytapoja.

4.1 Ensimmäinen iteraatio

Ensimmäisen iteraation aikana käytettiin aikaa tulevan ohjelmiston suunnitteluun ja vaatimusmäärittelyyn, sekä vanhan ohjelmiston analysointiin ja takaisinmallinnukseen. Vaatimusmäärittelyyn kuului myös vanhan Tremedia intra -ohjelmiston laajuuden ja toimintojen kartoittamisen. Vanhan ohjelmiston kuvaus ja laatuarviointi löytyy seuraavasta kappaleesta. Uuden ohjelmiston vaatimusmäärittely sisälsi nykyisen tietokannan rakennekaavion, sekä uudistetun ohjelmiston tietokannan suunnitelmakaavion. Ensimmäisen iteraation kesto oli noin neljätoista päivää.

4.1.1 Vanhan ohjelmiston kuvaus, analysointi sekä laatuarviointi

Tremedia Intra on web-pohjainen hallintojärjestelmä joka on kehitetty vuonna 2007. Ohjelmiston rakentamiseen on käytetty pääosin PHP kieltä. Lisäksi sen rakennuksessa on hyödynnetty JavaScriptiä, jQueryä ja Ajaxia.



Kuva 6. Vanhan Tremedia Intra - ohjelmiston etusivun ulkoasu (projekti ja asiakastiedot piilotettu kuvasta tietoturvasyistä)

Arviointia varten luotiin ohjelmiston arkkitehtuuri kuvas, jotta pystyttiin paremmin ymmärtämään nykyistä ohjelmistoa. Vanhasta ohjelmistosta oli saatavilla todella vähän dokumentaatiota, ja koska alkuperäinen kehittäjä ei ollut enää töissä yrityksessä, oli ohjelmiston toiminta myös epäselvää nykyisille ohjelmoijille. Dokumenttien puutteellisuuden vuoksi arkkitehtuurikuvausta varten jouduttiinkin tutkimaan paljon vanhaa ohjelmistoa koodi- ja tietokantatasolla.

Ohjelman arkkitehtuurikuvauksen perusteella pystyttiin paremmin arvioimaan mitä erilaisia muutoksia pystyttäisiin tekemään nykyisen ohjelmiston perusteella. Teknisestä kuvauksesta ja tietokantarakenne kaaviosta selviää nykyisen ohjelmiston arkkitehtuuriin liittyen tärkeitä rakenteellisia seikkoja, joiden avulla myös ohjelmiston laatuanalysointi oli helpompaa. Arkkitehtuuri- ja tietokantakuvaus toimi ohjelmiston takaisinmallinnuksena.

Takaisinmallintaminen ei ollut helppoa, sillä ohjelmistossa ei ollut selkeää rakennetta jota se olisi noudatellut. Siksipä takaisinmallintaminen pyrittiin tekemään melko karkealuontoisesti ja nopealla aikataululla., Takaisinmallinnus tietokannasta pystyttiin tekemään osittain automaattisesti käyttäen avuksi MySQL Workbench -ohjelmistoa. Vanhan ohjelmiston SQL tiedosto ladattiin ohjelmistoon, jolloin sen avulla pystyttiin luomaan tarkastelua helpottamia näkymiä (view).

Vanhalle Tremedia Intra -ohjelmistolle tehtiin ominaisuuksien kartoittaminen, jotta voitaisiin tämän avulla tehdä päätöksiä esimerkiksi niiden muutoksista. Lisäksi yritykselle oli tärkeää että uusi ohjelmisto tarjoaisi vähintään ne ominaisuudet kuin mitä vanha ohjelmisto sisälsi. Tärkeimmistä toiminnallisuuksista tehtiin yksityiskohtaisempi kuvaus. Myöskään ominaisuuksien kuvailuun tai liian yksityiskohtaiseen tarkasteluun, ei käytetty aikaa, sillä tarkentavaa tutkimusta tultaisiin tekemään myös kussakin iteraatiossa ominaisuuksien uudelleenrakentamisen yhteydessä. Toiminnallisuuksia kuvailtiin lähinnä niiden toimintojen kannalta ja esimerkiksi ohjelmistokoodia ei tarkasteltu tai takaisinmallinnettu lainkaan ominaisuuksista.

Vanhaa järjestelmää analysoitaessa selvitettiin myös mahdollisuus ohjelmistokoodin uudelleenkäytöstä. Uudelleenkäytöllä tarkoitetaan tässä tapauksessa koodin fyysistä siirtämistä vanhasta järjestelmästä uuteen järjestelmään. Analyysissä selvisi, että mitään ohjelmistossa olevaa moduulia eli osaa, ei voitaisi suoraa integroida osaksi uutta ohjelmistoa. Syynä voidaan nähdä vanhan ohjelmistokoodin huono rakenne ja sekavuus. Integroinnin seurauksena olisi jouduttu tekemään olemassa olevaan koodiin paljon muutoksia. Uudelleenkehitetyt ominaisuudet olisivat oletusarvoisesti paljon optimoidumpia.

Integrointikelpoista koodia pystyttäisiin kuitenkin kehityksen yhteydessä aina hakemaan ja tuomaan uuteen ohjelmistoon. Tarkempaa selvitystä ei ollut järkevää tehdä, sillä koodin integrointikelpoisuuden päätös kävisi kunkin ominaisuuden kohdalla nopeammin kehityksen aikana.

Analyysissä päädyttiin siihen tulokseen, että vanhan järjestelmän ohjelmistokoodista voitaisiin uudelleen käyttää erilaisia yksittäisiä toimintoja, kuten metodeja tai tyylejä käyttöön uuteen ohjelmistoon. Lisäksi erilaisia toimintomalleja, kuten kyselyiden optimoituja rakenteita, voitaisiin mahdollisuuksien mukaan hyödyntää projektissa.

Erityisen vaikeaa analyysissä oli pyrkiä vain karkeasti selvittämään ja takaisin mallintamaan vanhaa ohjelmistoa. Koska päätös uuden ohjelmiston rakentamisesta oli tehty, tarkoituksena ei ollut enää liian syvällisesti perehtyä vanhan ohjelmiston ohjelmakoodiin tai sen toimintojen monimuotoisuuteen, vaan pyrkiä vain löytämään tavoitteellisesti ne asiat mitä oikeasti hyödynnettäisiin.

Laatuarvioinnissa otettiin huomioon työntekijöiden lyhyt käyttäjäkysely (Liite 1) nykyisestä ohjelmistosta, kysely kosketti lähinnä ohjelmiston käyttöliittymää ja käytettävyyttä. Esitutkimus- ja määrittelyvaiheissa tehtiin tietokannan ja ohjelmistokoodin laatuarviointia. Lisäksi arvioitiin vanhan ohjelmiston ylläpidon puutteita (ylläpitoa vähentävät ja lisäävät laatutekijät). Laatuarvioinnin tuloksia kirjattiin jatkuvasti, suurimpina puutteina vanhassa ohjelmistossa nousi eteen seuraavat kohdat:

1. tietokannan rakenteen suunnitteluvirheet (laatu: toiminnallinen sopivuus).
2. ohjelmiston toiminnallisuuteen liittyvät suunnitteluvirheet (laatu: toiminnallinen sopivuus).
3. ohjelmistokoodin sekavuus ja epäloogisuus (laatu: ylläpidettävyys).
4. ylläpidettävyyden vaikeus ja hitaus (laatu: ylläpidettävyys).
5. dokumentaation puute (laatu: ylläpidettävyys).
6. suorituskyvyn puute (laatu: tehokkuus)

4.1.2 Vaatimusmäärittely

Koska ketterissä kehitysmenetelmissä pyritään ohjelmistoa rakentamaan osissa, myös vaatimusmäärittelyä tehtiin osissa. Vaatimusmäärittelylle luotiin siis aluksi runko, joka sisälsi uuden ohjelmiston karkean arkkitehtuuri-

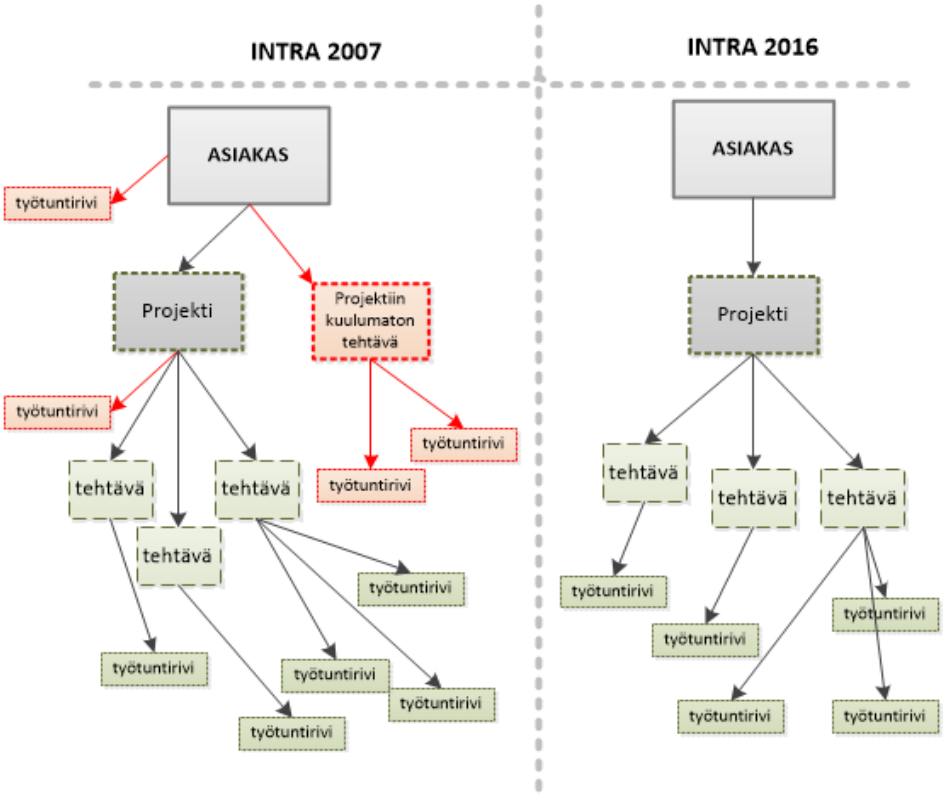
kuvauksen, karkean suunnitelman käyttöliittymästä sekä alustavan tietokanta suunnitelman. Kaikkien näiden luomiseen käytettiin takaisinmallinnuksessa saatuja tuloksia vanhasta ohjelmistosta.

Vaatimusmäärittelyn osaksi koottiin esitutkimuksessa selvinneet ohjelmiston rakentamisen pääkohdat, eli alustavat laatuvaatimukset. Pääkohdista selviää mm. mitä käytettävyyden periaatteita halutaan painottaa uudessa ohjelmistossa.

Lisäksi vaatimusmäärittelyn aikana koottiin useamman kohdan kysymyslista siitä miten nykyisen ohjelmiston toimintaa tullaan muuttamaan. Kysymyslistan tarkoituksena oli kyseenalaistaa nykyisen ohjelmiston toimintoja ja löytää ratkaisuja ohjelmiston epäkohtiin joita käyttäjät kohtaavat jokapäiväisessä ohjelmiston käytössä.

Vaatimusmäärittelystä ja alustavista arkkitehtuurikuvauksista pidettiin kehityspalaveri jonka mukaan tehtiin päätöksiä miten vaatimusmäärittelyn kysymyslistan kohtiin tultaisiin vastaamaan ja millaisia muutoksia olisi jo mahdollista tehdä ohjelmistoarkkitehtuurisuunnitelman perusteella. Kehityskokouksessa esimerkiksi vertailtiin sekä uuden, että vanhan ohjelmiston tietokanta-arkkitehtuuria ja suunniteltiin miten tietokantaa voitaisiin eheyttää nopeammaksi.

Kysymyslistan perusteella tehdyt päätökset uuden ohjelmiston toimintaan vaikuttavista asioista vaikuttivat myös yrityksen sen hetkisiin käytäntöihin. Koska päätöksiin kuului esimerkiksi projektien ja tehtävien kirjaamiseen liittyviä muutoksia, otettiin nämä käytännöt käyttöön jo nykyisen ohjelmiston kanssa. Koska päätettiin jatkossa tehtävien olevan aina riippuvaisia ohjelmiston projekteista, muutettiin projektien merkintäkäytäntöä nykyisessä ohjelmistossa niin, että työntekijöitä ohjattiin tekemään uudet tehtävät aina projektin alle. Asiakkaille luotiin yleensä vuositason projekti jonka alle käyttäjät pystyivät lisäämään esim. kuukautispainotteisen tehtävän johon tunnit merkittiin.



Kuva 7. Kysymyslistaan liittyvä muutos kuvattuna, jossa kyseenalaistettiin yksittäisten projektiin kuulumattomien tehtävien pysyminen käytössä. Lisäksi vanhasta ohjelmistosta hylättiin mahdollisuus lisätä työtuntirivejä suoraan projektin tai asiakkaan alle. Kun nämä päätökset oli tehty, myös vanhan ohjelmiston käytössä ohjeistettiin käyttäjille niiden toimintojen käyttämisen hylkääminen.

Vaatimusmäärittelyssä suunniteltiin myös ohjelmiston tulevaa visuaalista ilmettä ja käytettävyyteen liittyviä kohtia, kuten painikkeita. Visuaalisen ilmeen haluttiin myös olevan helposti muokattavissa, jolloin koko ohjelmiston rakenne on mietitty toimivan muutamalla päävärillä ja helposti vaihdettavilla kuvakkeilla. Visuaalisessa ilmeessä tärkeänä nähtiin myös se että tietueiden listautuminen sivustolle tehtäisiin mahdollisimman selkeäksi. Käyttöliittymän ja visuaalisen suunnittelussa otettiin huomioon erittäin paljon vanhan ohjelmiston laatu-arvioinnissa saadut tulokset.

Myös toimintopainikkeet ja logiikka haluttiin pitää mahdollisimman samankaltaisena kuin nykyisessä järjestelmässä, jolloin niiden käyttäminen olisi aina mahdollisimman samankaltaista. Käyttäjätestauksissa selvisi, että nykyisen järjestelmän ulkoasuun ei oltu tyytymättömiä, mutta käytettävyyteen, suorituskykyyn ja luotettavuuteen puolestaan oltiin. Esiin nousi myös se, että vanhan ohjelmiston käytössä arvostettiin sen yksinkertaisuutta. Käyttäjätestauksen tulokset löytyvät opinnäytetyön lopusta (Liite 1).

Ohjelmiston graafisen ilmeen ja käyttöliittymän suunnittelu pyrittiin tekemään useammassa osassa, jolloin koko ohjelmiston ilme kehittyi jokaisen iteraation aikana. Toimintoja ja ulkoasua suunniteltiin rautalankamalleissa, mutta varsinainen graafisen ilmeen kehittäminen tapahtui aina kunkin iteraation rakentamisen aikana.

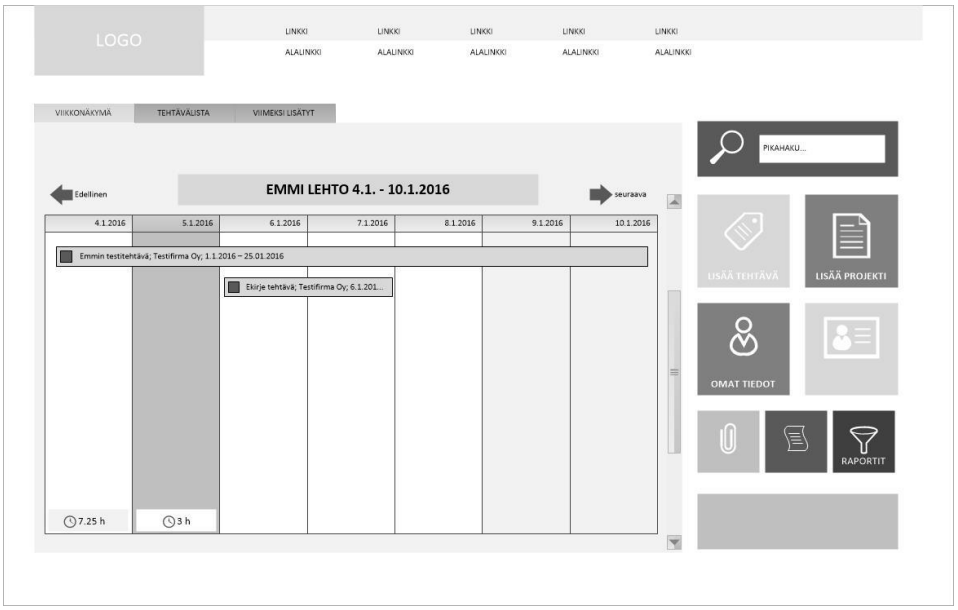
Ohjelmiston pohja koostuu useammasta Tremedian suunnittelemasta ja toteuttamasta ohjelmistosta. Esimerkiksi. kirjautumisen ja käyttäjähallinnan pohja koostuu mm. Tremedia CMS (julkaisujärjestelmä) ja Tremedia Verkkokauppaan tehdystä koodipohjasta, joissa on otettu valmiiksi huomioon mm. luokkien helppo rakentaminen ja hallinta. Koodipohja on toteutettu SQL:llä ja PHP:lla. Ensimmäisen iteraation aikana ei syntynyt varsinaista ohjelmakoodia, mutta saatiin projektin arkkitehtuurisuunnitelma valmiiksi jonka mukaan järjestelmää alettaisiin ohjelmoida.

4.2 Toinen iteraatio

Toisen iteraation aikana alettiin tehdä ohjelmakoodia ensimmäisen iteraation määrittelyiden pohjalta. Vaikka määrittelyä oli tehty jo aikaisemmassa iteraatiossa, suunnittelu jatkui kuitenkin ominaisuuskohtaisesti. Toisen iteraation kesto oli noin neljätoista päivää.

4.2.1 Käyttäjät, käyttäjähallinta ja kirjautuminen

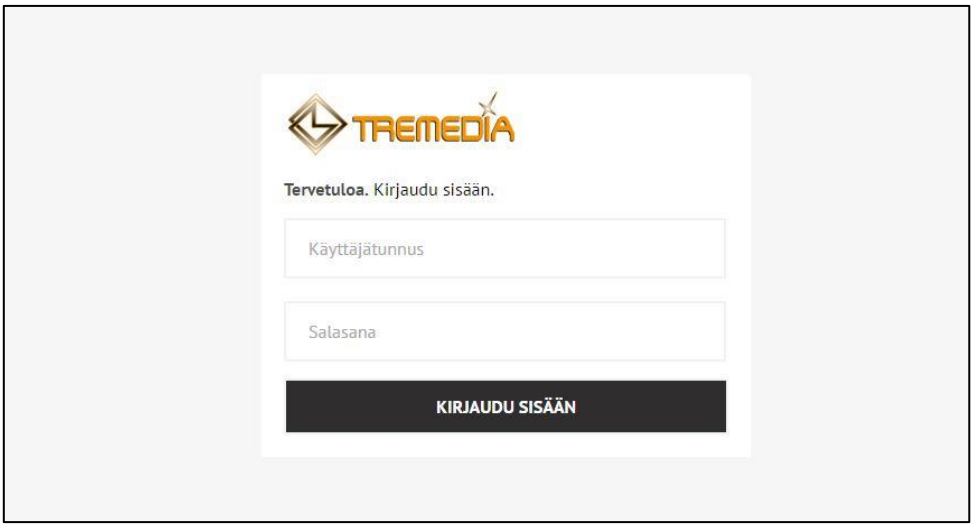
Ohjelmiston toteutus alkoi kirjautumisnäkymän ja käyttäjähallinnan rautalankamallin rakentamisella. Lisäksi rautalankamalli tehtiin alustavasti myös etusivusta, jonka perusteella pystyttiin tekemään alustavaa graafista suunnittelua koko ohjelmistolle.



Kuva 8. Ensimmäinen rautalankaversio ohjelmiston etusivusta. Etusivun olomuotoa muokattiin ensin rautalankamallin suunnitelmaan ja lopulta toteutuksen aikana inkrementaalisesti pysyvään olomuotoonsa.

Rautalankamallien perusteella lähdettiin rakentamaan ohjelmiston ensimmäistä osaa. Ohjelmistoon rakennettiin ensin käyttäjiin ja kirjautumiseen liittyvät tietokantataulut ja tehtiin indeksointi näiden taulujen välille. Taulujen rakentamisen yhteydessä luotiin myös tarvittavat luokat ja näiden pohjafunktiot eli metodit kullekin luokalle. Käyttäjähallinnan ja kirjautumisen koodipohjalla käytettiin Tremedian Verkkokaupan muutamaa eri portaalia,

esimerkiksi portaalia jota käytetään tilauksien hallintaan. Koska portaalit ovat rakennettu hyvällä tekniikalla, periytettiin näiden moduuliliitoksien yhteydessä koko ohjelmiston pohjalle tarkoitettu koodaus- ja dokumentointitavat.



Kuva 9. Toteutunut kirjautumisnäkymä ohjelmiston betaversiossa

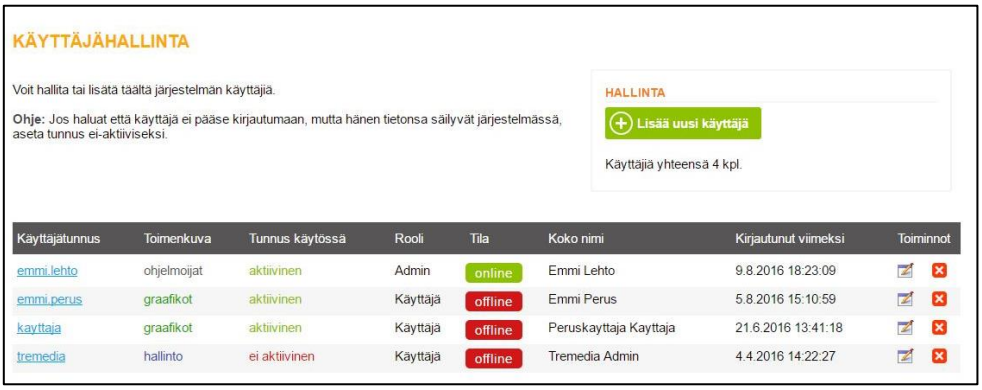
Tässä iteraatiossa ei uudelleen käytetty vanhan Tremedia Intran ohjelmiston koodia tai scriptejä, sillä todettiin uudempien portaalien kirjautumisen ja käyttäjähallinnan koodien olevan paljon paremmin tietoturvatestattuja, uudemmalla tekniikalla rakennettuja ja paremmin dokumentoituja.

Kirjautumisen ja käyttäjähallinnan rakentamisen yhteydessä tehtiin myös muuta pohjatyötä ohjelmistoon. Esimerkiksi ohjelmiston kirjastomääritykset, rakennemäärityksen ja perinteiset metodit joita ohjelmistossa tullaan jatkuvasti käyttämään. Tähän iteraatioon kuului myös luonnollisesti tyylien (CSS) määrittäminen ja sivuston alustavan ulkoasun rakentaminen. Tyylien rakentaminen aloitettiin lähes tyhjästä, sillä ohjelmoija tuottaa tehokkaasti ulkomuistista tyylikoodia. Osaa tyylikoodeista uudelleen käytettiin eri portaaleista rakentamisen edetessä.

Frameworkin tai vanhojen tyylien käyttäminen järjestelmään olisi ollut huomattavasti hitaampaa kuin vain tehdä kullekin elementille omat tyylinsä. Vaikka ohjelmistoa ei ole tarkoitus toteuttaa responsiivisena (kaikilla päätelaitteilla toimivana), kuitenkin tyylien rakentamisessa otettiin huomioon, että ohjelmisto näkyisi ainakin isommilla ja pienemmillä näytöillä hyvältä ja että kaikki toiminnot olisivat saatavilla. Myöskään responsiivisuutta ei haluttaisi kokonaan pois sulkea kehityksestä, sillä ohjelmiston käyttäminen esimerkiksi tabletilla joskus tulevaisuudessa haluttaisiin pitää mahdollisena vaihtoehtona.

Pohjatyöhön liittyviä metodeja ovat esimerkiksi tietoturvaan liittyvät XSS-funktiot joilla tietueita voidaan tulostaessa käydä läpi, muuttaa ja tarkastaa. Tämä on tärkeää sillä, kaikki käyttäjän antamat syötteet joissa tietoa viedään tietokantaan, täytyy käydä läpi. Vahingollisten syötteiden avulla voi olla mahdollista esimerkiksi vahingoittaa koko ohjelmiston tietokantaa.

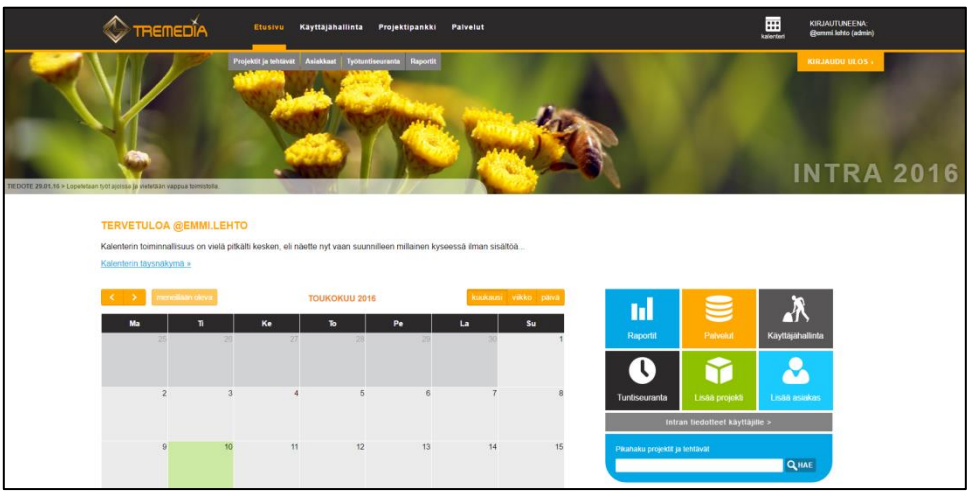
Johtoryhmälle järjestetyssä kehityskokouksessa tuli esiin muutamia haluttuja toimintoja jotka saatiin todella nopeasti toteutettua uuteen ohjelmistoon. Näitä toimintoja oli mm. IP -osoitteiden ja aikaleiman tallentaminen kirjautumishistoriana.



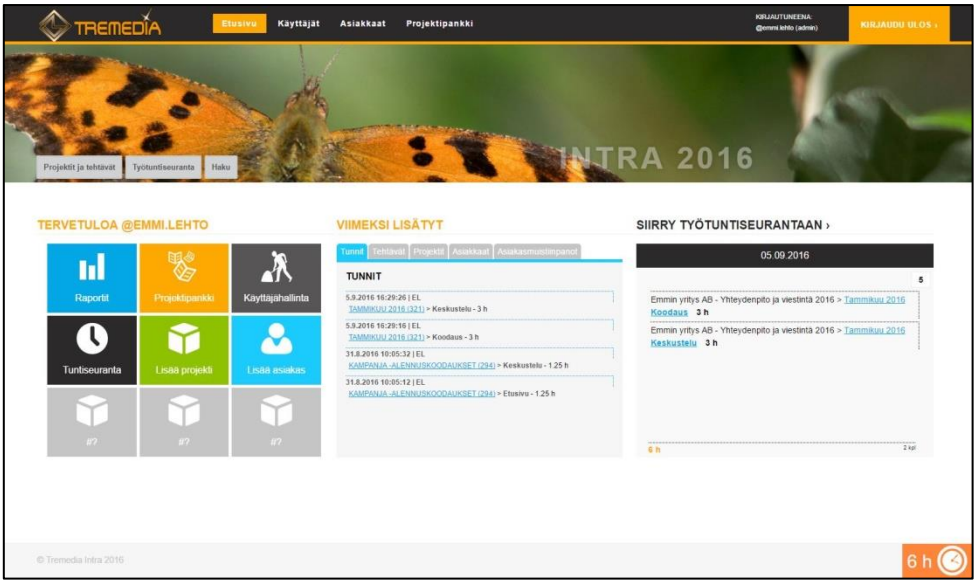
Kuva 10. Toteutunut käyttäjähallinta -sivu ohjelmiston betaversiossa

Kirjautumiselle tehtiin perinteinen muutaman kohdan manuaalinen tietoturvatestaus, mutta lopullinen testaus tehtäisiin betaversion julkistamisen jälkeen, jolloin ohjelmiston testaukseen käytettäisiin hyödyksi verkkosovellusten testaamiseen käytettävää Acunetix -ohjelmistoa.

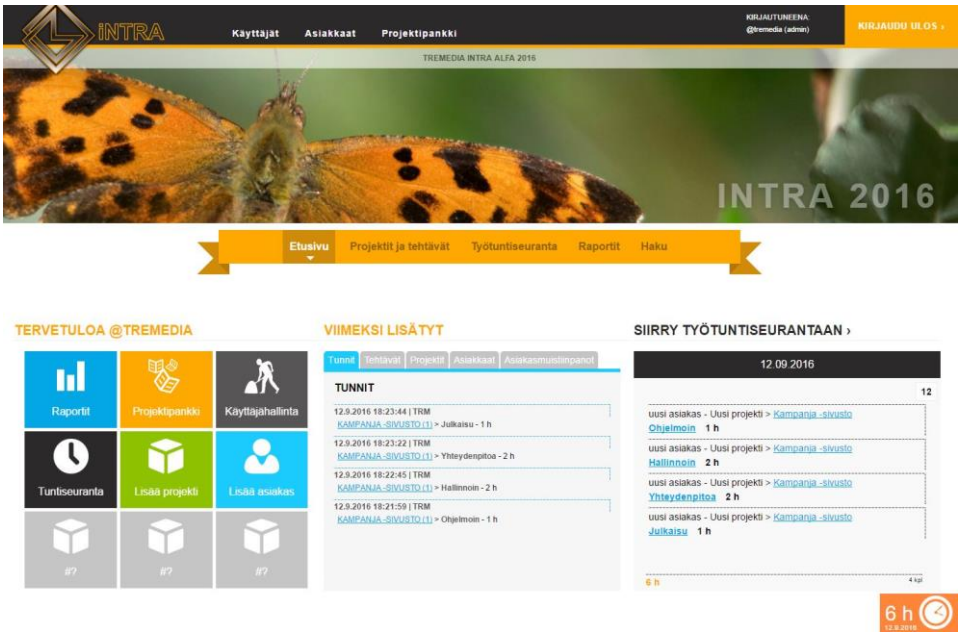
Myös etusivua alettiin kehittää rautalankamallin mukaisesti eteenpäin, etusivuun tehtiin muutoksia matkan varrella, joiden uskottiin parantavat näkymän käytettävyyttä. Alla olevissa kuvissa 11 ja 13 näkyy selkeä ero, miten ohjelmiston etusivu on kokenut käytettävyyttä parantavia muutoksia.



Kuva 11. Etusivun prototyyppiversio, jossa nähtävillä alun perin käyttöön suunniteltu fullcalendar.js -kirjasto.



Kuva 12. Etusivun alfaversio, jossa yhdistyy alkuperäisen rautalankasuunnitelman mukaisia ominaisuuksia ja piirteitä, mutta selkeämmin jäsenneltynä.



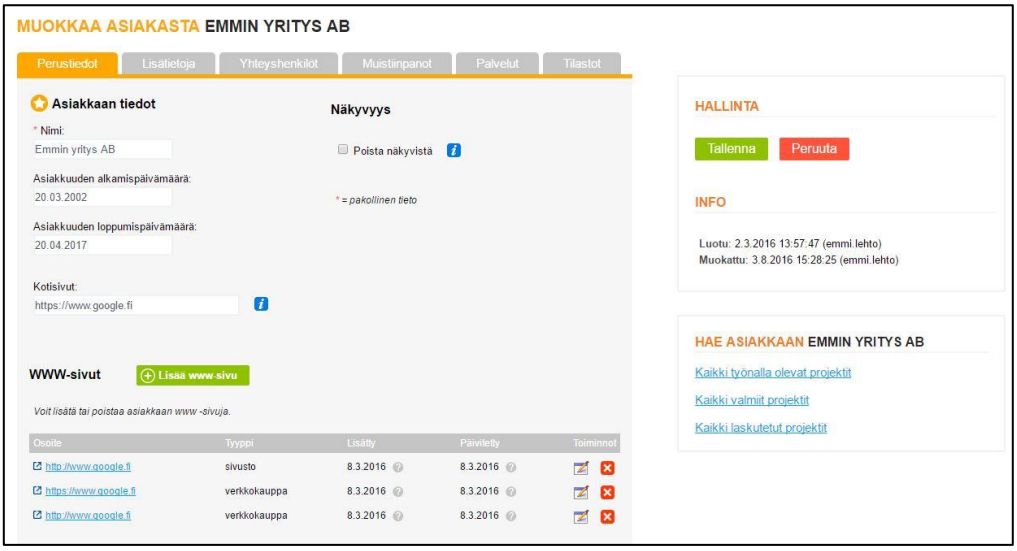
Kuva 13. Ohjelmiston etusivun betaversio. Etusivun kehityskaari kuvaa hyvin sitä miten kehittäminen on tapahtunut iteraatioiden välissä ja suuriakin muutoksia karkeaan alkuperäissuunnitelmaan on tehty.

4.2.2 Asiakashallinta

Asiakkaat ovat suuressa osassa ohjelmistossa, koska se on tärkeä osa tietojen ryhmittämistä. Asiakkaiden osuus myös tietokantarakenteessa on tärkeä koska se määrittelee paljon koko ohjelmiston toimintalogiikan rakentamista.

Asiakkaiden, projektien ja tehtävien suhdetta voidaan kuvata kuvalla 15. Käytännössä voidaan ajatella, että asiakkaat ovat korkein määrääjä ja projektit kuuluvat aina asiakkaalle ja tehtävät aina projektille. Käytännössä siis järjestelmässä ei voi olla projektia jolla ei ole asiakasta tai tehtävää joka ei kuulu projektiin. Tämä ajattelutapa poikkeaa edellisestä Tremedia Intra -ohjelmistosta sillä, että tehtävät voitiin merkitä myös yksittäisinä, eli niiden ei ollut pakko kuulua aina projektiin. Muutoksen päättäminen tehtiin ohjelmiston esitutkimusta ja vaatimusmäärittelyä tehdessä.

Asiakashallinnan ohjelmointi tapahtui kopioimalla käyttäjähallinnan tietokantataulut, tiedostot ja luokat, sekä muokkaamalla niitä asiakashallinnassa käytetyille tiedoille. Asiakashallinnassa käytetyt saraketiedot kopioitiin nykyisestä järjestelmästä ja niihin lisättiin myös tarvittavia kenttiä, kuten mahdollisuus merkitä asiakkuuden loppumisaika.



Kuva 14. Toteutunut asiakkaan muokkaus -sivu ohjelmiston betaversiossa.

Johtoryhmän kehityskokouksessa haluttuja toiminnallisuuksia oli esimerkiksi yhteyshenkilöiden merkitseminen asiakkaalle, www-osoitteiden kerääminen ja muistiinpanojen kirjaaminen kunkin asiakkaan kohdalle. Lisäksi esiin nousivat myös tilastot -välilehden rakentaminen, jossa voitaisiin analysoida tarkemmin asiakkaan projektitietoja esimerkiksi koko vuodelta.

4.3 Kolmas ja neljäs iteraatio

Kolmannen ja neljännen iteraation kohdalla keskityttiin siihen mikä on ohjelmiston kannalta merkittävin osa: projektien ja työtehtävien väliset liittymät sekä niiden toiminnot. Iteraatioihin käytettiin yhteensä noin 28 päivää.

Näihin iteraatioihin liittyen tehtiin huomattavasti eniten suunnittelua. Tavoitteeksi nousi se, että käyttäjä pystyisi mahdollisimman paljon määrittelemään asioita jotka säilyisivät koko istunnon lisäksi myös välimuistissa tai tallennettuna tietokantaan. Nämä käyttäjäkohtaiset määritteet helpottaisivat sitä, että kukin käyttäjä pystyisi pitämään vain ne projektit näkyvillä nopeasti mitkä heitä kiinnostavat.

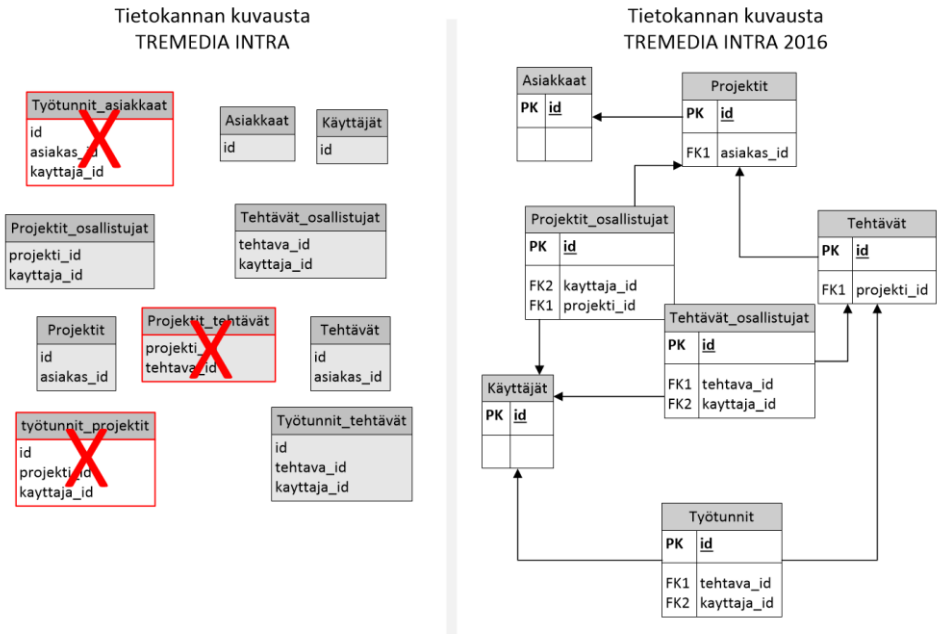
4.3.1 Projektit ja työtehtävät

Projektien ja työtehtävien suunnittelu alkoi johtoryhmän kehityskokouksessa, jossa mietittiin miten voitaisiin parantaa projektien listautumista helposti ja nopeasti. Palaverin aikana nousi esiin monia asioita, esimerkiksi kuinka työntekijä etsii yleensä järjestelmästä niitä projekteja joihin hän on osallisena ja mitä hän tekee kunakin viikkona.

Kokouksessa nousi esiin mm. seuraavat kehitysajatukset: Valikkoihin haluttaisiin käyttöön verkkokauppatyylinen valintapainike, josta voitaisiin valita ne projektit mitä ollaan rakentamassa. Vertailuna tässä verkkokaupan suodatuspainikkeet josta näytettävät filtterit voidaan valita näytettäväksi. Tämä rajaisi tehokkaasti haettavan datan määrää, jolloin ohjelmiston suorituskyky nousisi (ohjelmiston laatuvaatimukset).

Suodatusvaihtoehtoja haluttaisiin monipuolistaa kaikissa näkymissä, esimerkiksi vain kuluvan vuoden projektien näyttäminen listauksessa. Tavoitteena olisi myös rakentaa ominaisuus, jonka mukaan käyttäjä pystyy tallentamaan omat asetuksensa oletukseksi. Tällöin tehdyt suodatusvalinnat säilyvät kunnes ne korvataan uusilla. Tämän lisäksi istuntoon tallennettu suodatus joka säilyy koko istunnon ajan tai kun käyttäjä kirjautuu ulos. Lisäksi tavoitteena olisi hyödyntää myös selaimen välimuistia, jolloin ohjelmiston suorituskyky nousisi.

Ensimmäiseksi suunniteltiin asiakkaiden, projektien ja työtehtävien välistä suhdetta tietokannassa. Vanhassa järjestelmässä oli tullut näiden kohdalla esiin paljon ongelmia liittyen tietojen järjestelyyn. Rakennetta tulikin uudistaa ja lisäksi koko ajattelumallia muutettiin jo ensimmäisten kehitysideoiden mukaisesti. Yksittäisten, projektiin kuulumattomien, tehtävien poistaminen tietokanta rakenteesta helpotti sitä huomattavasti.



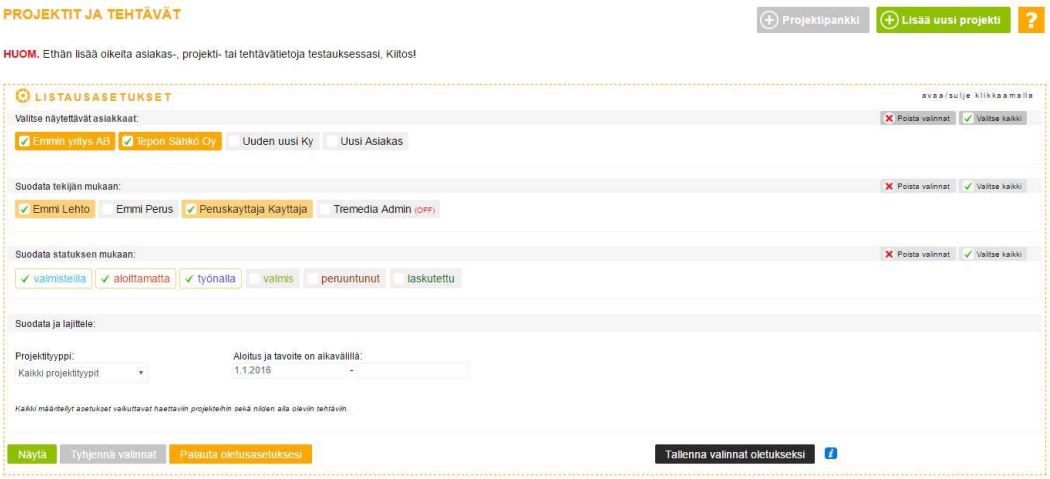
Kuva 15. Tietokannan rakennetta yksinkertaistettiin uuteen ohjelmistoon ja käytettiin pää- ja viiteavaimia. Punaisella merkityt taulut poistuivat kokonaan käytöstä.

Ohjelmoinnissa pyrittiin seuraamaan tietokannan määrittelemän rakenteen mukaisesti sekä logiikkaa, että käyttöliittymää. Esimerkiksi: koska projekti kuuluu aina asiakkaalle, oli järkevää laittaa asiakas ensimmäiseksi suodatustekijäksi. Koska tehtävät kuuluvat aina projektille, oli järkevää niputtaa tehtävät suoraan projektin alle.

Projektien ja tehtävien kuvausta jatkettiin niin että tehtiin käyttöliittymästä rautalankamalli, joka esiteltiin projektiryhmälle. Tämän jälkeen aloitettiin pohjatoimintojen, luokkien ja olioiden rakentaminen, sekä kyselyrakenteen suunnittelu. Tässä vaiheessa palattiin lähes päivittäin suunnittelemaan ominaisuuksien tarkennuksia ohjelmoinnin yhteydessä. Myös takaisinmallinnusta hyödynnettiin jatkuvasti, jolloin tietokannan optimointi oli tehokkaampaa ja epäkohdat pystyttiin paikallistamaan vanhasta ohjelmistosta.

Ohjelmoinnin puolella tärkeäksi osaksi muodostui ns. projektien ja tehtävien listausnäkymän listausasetukset -paneeli. Paneelin tehtävänä oli suodattaa haluttuja näytettäviä tuloksia näkymässä. Paneelia suunniteltiin pitkään rautalankamallina, joka esiteltiin ohjausryhmälle ja siihen tehtiin palautteen mukaisia korjauksia. Alla olevissa kappaleissa on selvennetty lukijalle paneelin toimintaa.

Kuva 16. Toteutunut projektien ja tehtävien listausnäkymä suodatuspaneeli ohjelmiston betaversiossa. (Paneeli kiinni)



Kuva 17. Toteutunut projektien ja tehtävien listausnäkymän suodatuspaneeli ohjelmiston kehitysversiossa. (Paneeli auki)

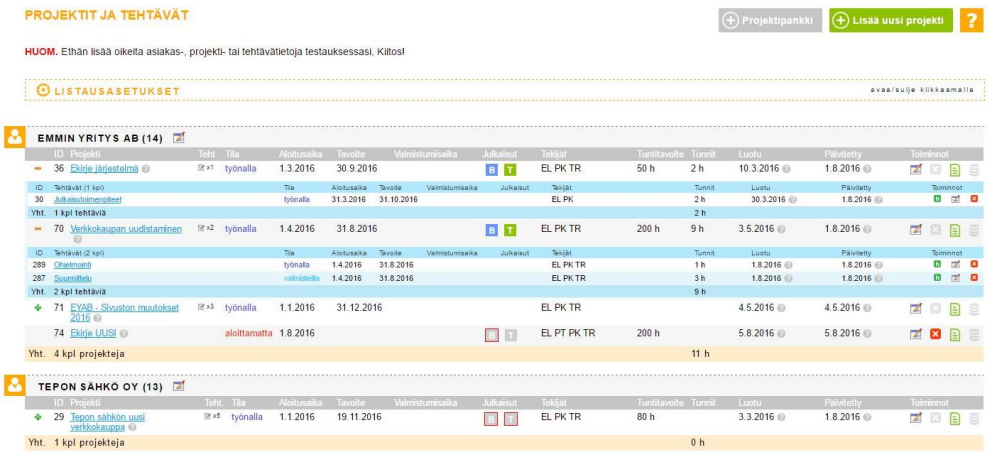
Paneelissa valitaan ensin asiakkaat joita halutaan tarkastella. Tähän listautuu vain ne asiakkaat joilla on aktiivisia projekteja tietokannassa. Tämän jälkeen voidaan valita seuraavista suodatuskohdista halutut asiat. Admin

puolen käyttäjät pystyvät myös suodattamaan käyttäjien mukaisesti projekteja. Lisäksi voidaan valita suodatuskriteereiksi status, projektityyppi, sekä projektin aloitus ja lopetusajankohta. Paneelin pystyy myös sulkemaan, jolloin se vie vähemmän tilaa näkymästä. Sulkeminen ja avaaminen tallentuvat aina myös evästeisiin, jolloin paneeli ei avaudu vaikka sivua päivitetään tai käyttäjä kirjautuu ulos järjestelmästä. Tällöin käyttäjää ei häiritse paneeli ja hän voi itse valita milloin haluaa muuttaa suodatusehtojaan.

Kun haku on suoritettu ”Näytä” -painikkeesta, se tallentuu istuntoon. Tulokset listautuvat paneelin alapuolelle. Paneelissa voidaan myös tyhjentää kaikki asetukset yhdellä painikkeella.

”Tallenna valinnat oletukseksi” – painike vie määritellyt asetukset suoraa tietokantaan, josta ne haetaan aina kun käyttäjä kirjautuu uudelleen ohjelmistoon eli luo täyden uuden istunnon. Välissä käyttäjä voi tehdä muita tarpeellisia hakua ja tämän jälkeen palauttaa ”Palauta oletusasetuksesi” -painikkeen avulla jälleen tallennetut asetukset.

Oletushaun tekeminen on perusteltua monestakin syystä, ensinnä käyttäjä todennäköisesti toistaa samaa hakua tiettyjen asiakkaiden kohdalla päivittäin tai jopa kuukausittain. Esimerkiksi jos käyttäjän työlistalla on viikolla kolmen eri asiakkaan töitä, hän voi hakea vain niiden kolmen asiakkaan ”työn alla” - statuksella olevat projektit.



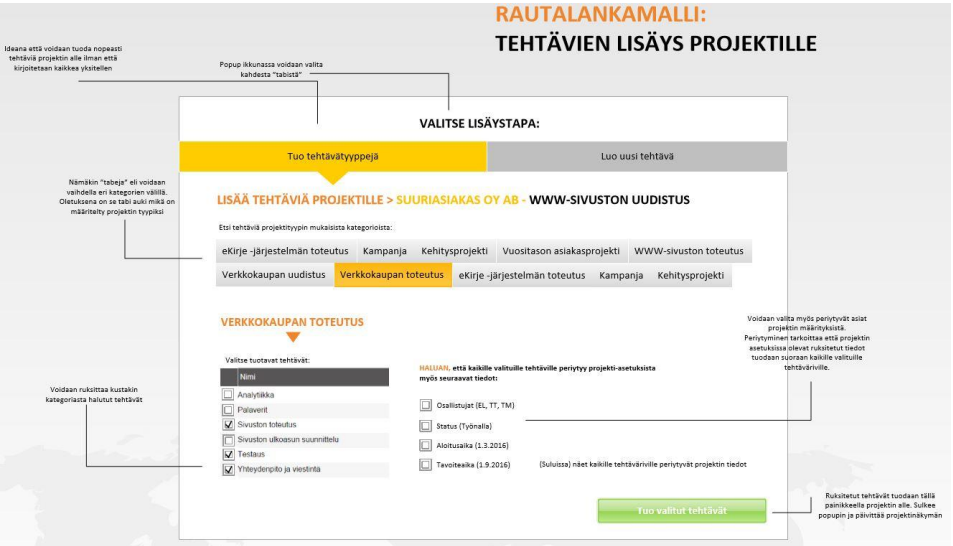
Kuva 18. Kun haku on suoritettu suodatuspaneelin mukaisesti, tulokset listautuivat sen alle asiakkaan mukaisesti allekkain.

Kuva 19. Projektipankissa pystytään lisäämään projektityyppejä ja niiden alle tehtävätyyppejä. Nämä nopeuttavat projektien suodattamista ja tehtävien luomista.

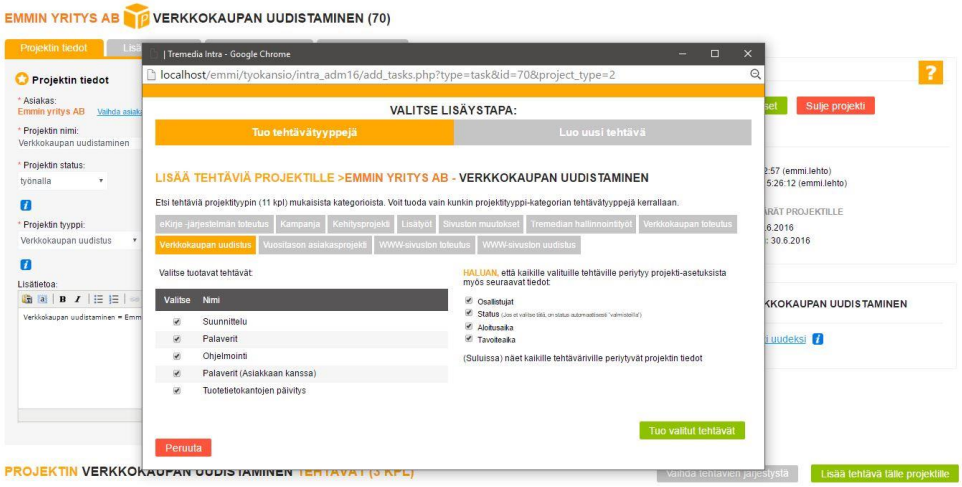
Erilaisilla kuvakkeilla on pyritty vähentämään muistikuormaa projektien ja tehtävien hallinnassa. Kuvakkeista tulee esiin lyhyet selitteet kun niiden päälle viedään hiiri. Lisäksi harmaat kysymysmerkkikuvakkeet kertovat enemmän tietoa kyseisestä kentästä kun sen päälle viedään hiiri. Esimerkiksi luotu -arvon yhteydessä olevaa lisätietoa on kuka on luonut ja kellonaika luonnille.

Projektityyppejä hallitaan projektipankista. Projektityyppi on projektilla pakollinen ja sen määritteleminen helpottaa projektien suodatusta ja lajittelua. Myöhemmin projektin tyypillä saattaa olla myös tilastoissa arvoa. Sen mukaan voidaan analysoida, minkälaisia projekteja tietyille asiakkaille tehdään ja kuinka paljon jne.

Projektityypeille tehtävätyyppien lisääminen helpottaa puolestaan projektin tehtävän luomista. On todettu että samat tehtävät toistuvat samanlaisissa projekteissa, jolloin niiden liittäminen projektiin tulisi olla ainakin osittain automatisoitua.



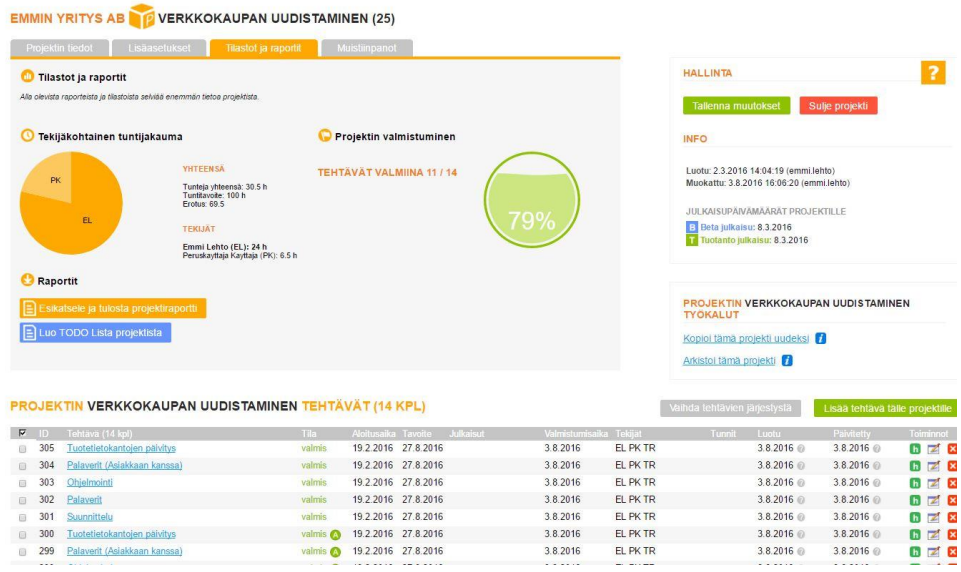
Kuva 20. Rautalankamalli ja toimintojen esittely tehtävien lisäyksestä projekteille.



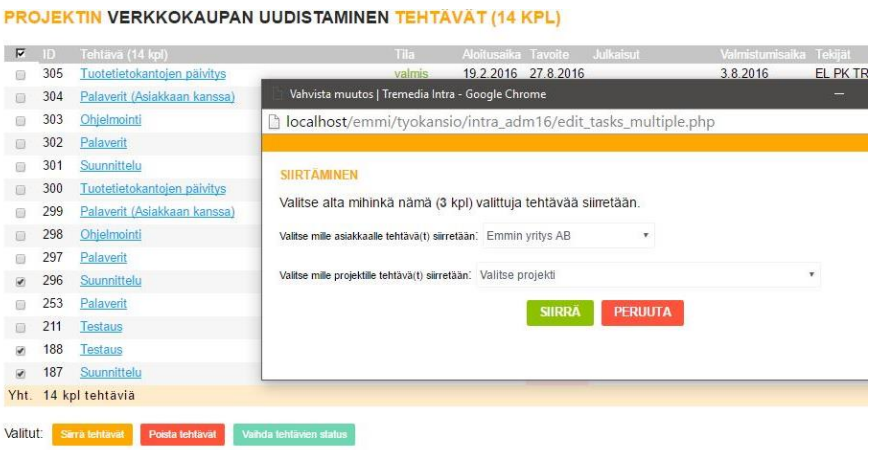
Kuva 21. Toteutunut tehtävän lisäys ikkuna. Kun projektille tuodaan uusia tehtäviä, tämä käy nopeasti projektityypin mukaisesti hakemalla sille kuuluvat tehtävätyypit ja halutessaan käyttäjä voi periyttää niille projektilta esimerkiksi. osallistujat

Tarvittaessa kun projektille tuodaan tehtävätyyppejä, eli tehtäviä, voidaan myös määrittää mitä ominaisuuksia tuodaan projektilta. Periytyviä ominaisuuksia ovat esimerkiksi osallistujat, status, aloitus- ja tavoiteaika. Lisäksi voidaan tuoda tehtäviä myös toiselta projektityypiltä tai tarvittaessa luoda uusi tehtävä täysin omilla asetuksilla.

Projektille pystytään määrittelemään myös lisäasetuksia, kuten julkaisutavoitteet ja varsinaiset julkaisupäivät (beta- ja tuotanto) ja lisäksi tuntitavoitteita. Muita ominaisuuksia on esimerkiksi projektin kopioiminen uudeksi projektiksi tai muistiinpanojen lisääminen projektille.
Projektien tilastot ja raportit osiosta pystytään luomaan erilaisia raportteja, lisäksi nähdään datavisualisoituna esimerkiksi projektin tekijäkohtainen tuntijakauma, sekä projektin valmistumisprosentti.



Kuva 22. Projektin tilastot ja raportit – välilehti ohjelmiston betaversiossa

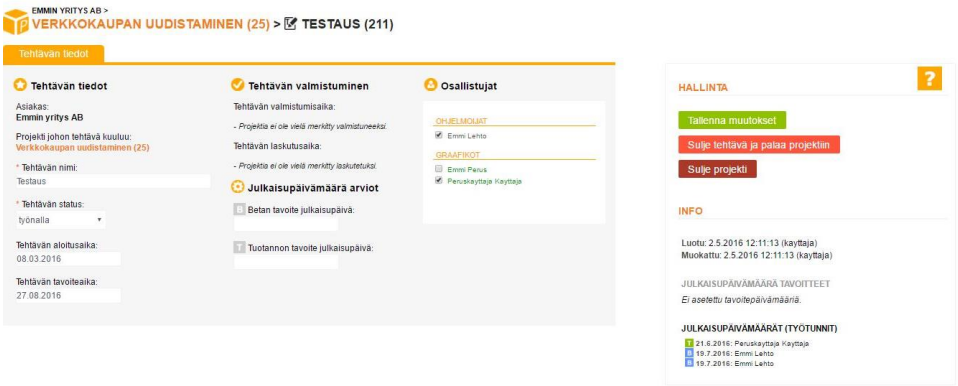


Kuva 23. Tehtävien valitseminen ja ”Siirrä tehtävät” -ryhmätoiminnon käyttö.

Projektin tehtävät listautuvat projektin asetuksien alle. Tehtäviä pystytään muokkaamaan suoraa listanäkymästä. Myös tuntimerkintöjä voidaan lisätä suoraan listanäkymistä kullekin tehtävälle. Ryhmätoimintojen avulla voidaan siirtää vaikka neljä työtehtävää työtunteineen ja muine tietoineen toiselle projektille.

Lisäksi tehtäviä voidaan järjestellä kunkin projektin alla ”drag and drop” –toiminnolla. Tämä tarkoittaa että järjestystä voidaan vaihtaa tarttumalla tehtävän nimeen ja pudottamalla se haluamaansa kohtaan listassa. Tämä toiminto on hyvä esimerkki ohjelmistokoodin uudelleenkäytöstä. Ohjelmistokoodi on uudelleenkäytetty Tremedia Verkkokaupan hallinnan tuotteiden järjestelystä.

Tehtävien muokkaaminen ja hallitseminen noudattaa pitkälti samaa kaavaa kuin projektien hallinta. Tehtäviä kuitenkin pystytään siirtämään projektilta tai jopa asiakkaalta toiselle, siirrossa otetaan myös aina huomioon tehtävälle lisätyt tuntimerkinnät.



Kuva 24. Tehtävien muokkaaminen ohjelmiston betaversiossa

4.4 Viides ja kuudes iteraatio

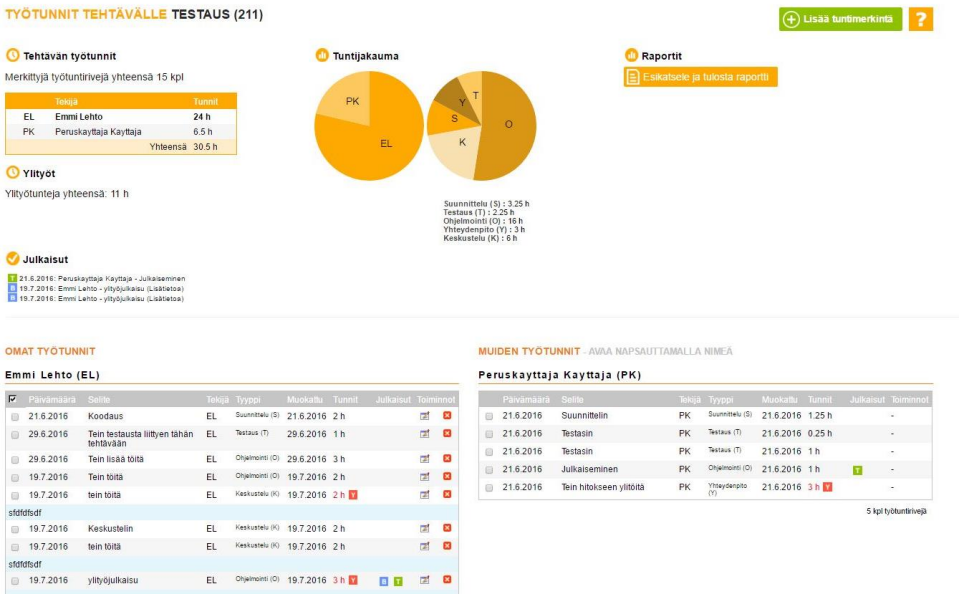
Kun projektien ja tehtävien rakenne, lista ja muokkausnäkymät oli rakennettu, oli mahdollista alkaa suunnittelemaan työtuntien näkyvyyttä ja hallintaa ohjelmistoon. Työtunteihin liittyviin iteraatioihin käytettiin yhteensä noin 28 päivää.

4.4.1 Työtunnit

Työtuntien rakentaminen aloitettiin tietokannan taulujen suunnittelulla ja rakentamisella. Avuksi käytettiin vanhan tietokannan takaisinmallinnuksen piirroksia, joiden perusteella pystyttiin toteuttamaan melko samanlainen tietokantarakenne. Seuraavaksi ohjelmointiin siihen liittyvät luokat ja oliorakenteet. Työtuntien lisäämisestä tehtiin myös muutama rautalankamalli, joissa käytettiin avuksi vanhan ohjelmiston takaisinmallinnusta ja vanhan käyttöliittymän arviointia.

Työtuntien tarkastelu tapahtuu tarkastelemalla tehtävän tietoja. Tehtävän tietojen alle tulee näkyviin aina sille kirjatut työtunnit. Työtunnit ovat jaettu kirjautuneen käyttäjän omiin työtunteihin ja muiden työntekijöiden työtunteihin.

Työtuntien yhteyteen haluttiin tuoda tuntien jakautumiseen liittyvää visualisoitua dataa, ja lisäksi mahdollisuus luoda tehtävä ja työtuntikohtainen raportti. Työtuntimerkintöjä pystyy myös muokkaamaan, poistamaan tai siirtämään eri tehtävien välillä.



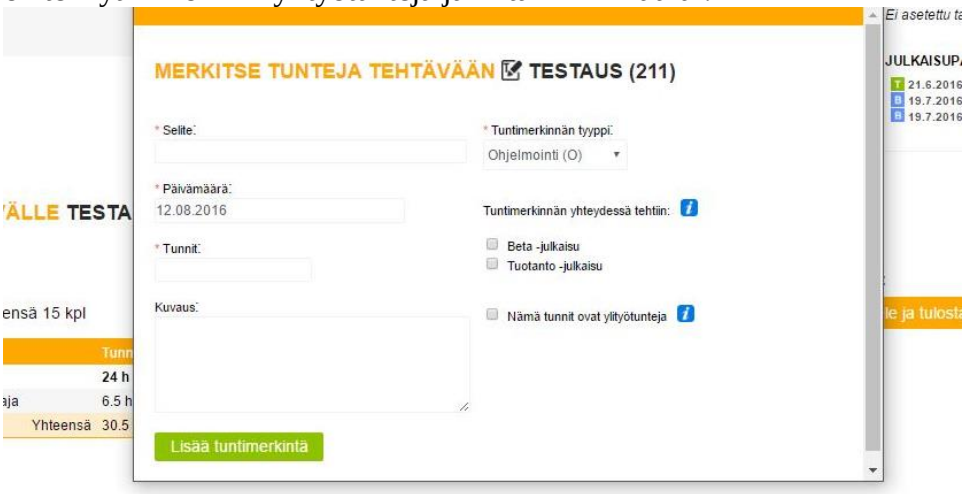
Kuva 25. Työtuntien listautuminen tehtävässä

Poiketen vanhan ohjelmistojen tuntidatan lisäämisestä, uudessa ohjelmistossa on ennalta valitut tuntimerkinnän tyypit, jonka mukaan luotava tuntirivi täytyy kirjata johonkin tyyppiin. Tämä auttaa työtuntien

lajittelussa ja analysoinnissa ohjelmistossa. Tuntimerkinnän tyypin oletusvalinta määräytyy sen mukaan, että mihin käyttäjäryhmään kirjautunut käyttäjä kuuluu. Hänen kuuluessaan esimerkiksi graafikoihin, oletuksena valikkoon tulee ”suunnittelu (S)”, ohjelmoijalla puolestaan ”Ohjelmointi (O)”.

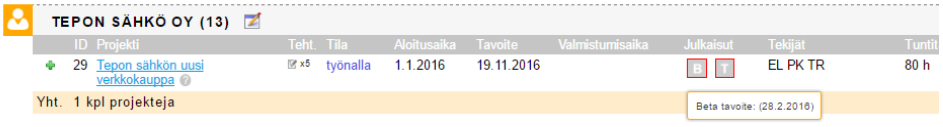
Tuntimerkintään pystytään merkitsemään tieto myös julkaisusta, eli jos työtunnin yhteydessä on tehty beta- tai tuotantojulkaisu, tämä tieto peritytyy tottakai myös tehtävälle ja projektille. Julkaisuajankohtia on mahdollista ohjelmistossa merkitä useampi kuhunkin työtehtävään. Tällöin pystytään paremmin analysoimaan tehtävien ja koko projektin elinkaarta kokonaisuudessa.

Ylityöksi työtuntirivin merkitseminen voi olla tärkeää, jos esimerkiksi projektissa on määritelty asiakkaalle erillinen kustannus tehtäville ylityötunneille. Merkintä tukee siis laskutusta ja näyttää samalla että kuka on tehnyt milloinkin ylityötunteja ja mitä niihin kuului.



Kuva 26. Työtuntien lisääminen tehtävälle.

Ohjelmistossa on pyritty käyttämään paljon erilaisia indikaattoreita, jotka kuvaavat esimerkiksi julkaisutavoitteiden myöhästymistä, projektin tavoiteajan ylittämistä ym.



Kuva 27. Projektin julkaisu- ja betatavoitteet on ohitettu ajallisesti, indikaattori osoittaa punaisella, että projektiin tulee kiinnittää huomiota esimerkiksi uudelleen aikatauluttamalla.

4.4.2 Kalenterinäkymät

Kalenterinäkymissä toiveena olivat että ohjelmistoon tulisi ainakin seuraavat näkymät: päivä-, viikko- ja kuukausinäkymät. Alun perin oli tarkoituksena hyödyntää tehtäviä ja projekteja kalenterissa, jolloin kukin henkilö

voisi aina personoida itselleen tehtävien mukaisen työlistan ja näin määrittää itselleen aikajanalle kalenteriin mitä tekee minäkin päivänä. Tähän löydettiinkin jo sopiva vapaasti käytettävä JavaScript -pohjainen kirjasto, jonka avulla saatiin ensimmäinen prototyyppi ohjelmoitua.

Ominaisuuksien kehittämiseen olisi kuitenkin kulunut liikaa aikaa koska kirjasto oli kuitenkin vain pohja jossa ominaisuuksia olisi pitänyt kehittää ja integroida muuhun ohjelmistoon. Tämä olisi vaatinut huomattavasti enemmän aikaa, joten lopulta alettiin etsimään vastaavaan hallintaan sopivia muita ratkaisuita. Lopulta päätettiin käyttää jo olemassa olevaa kaikille tuttua ohjelmistoa ja sen kalenteria, Microsoft Outlookia. Jokaiselle työntekijälle luotiin oma kalenteri, johon he voisivat itse merkitä ainakin viikoittain minä päivinä työstävät mitäkin projekteja.

Tämän yhteydessä nousi esiin myös mahdollisuus, jossa Outlook kalenteri pystyttäisiin REST API:n kautta tuomaan web-ohjelmistoon näkyviin. Office 365 tarjoaa mahdollisuuden lukea ja kirjoittaa kalentereihin verkkosovelluksesta yhdistämällä. Tämä tarkoittaa sitä että Intrasta voitaisiin jatkossa hallita myös Outlookiin perustettuja tekijäkohtaisia työlistakalentereita. Koska tämä ei kuulunut alkuperäiseen suunnitelmaan, siirrettiin tämän toteutus ohjelmiston kehityslistalle.

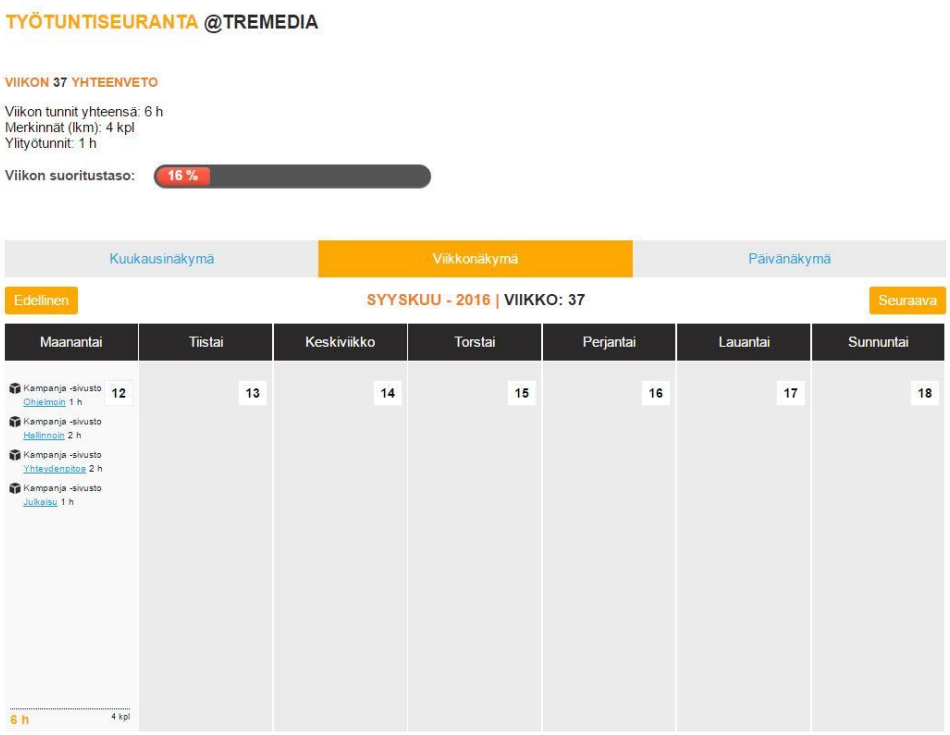
Ohjelmistoon haluttiin kuitenkin tuoda päiväkohtaiset tuntimerkinnät näkyviin. Vanhassa ohjelmistossa oli mahdollista nähdä sekä päivä-, että viikkokohtaisesti tehdyt työtunnit ja selaamaan kalenteria myös aiemmilta kuukausilta, viikoilta tai päiviltä.

Kuva 28. Vanhan ohjelmiston viikkonäkymä. Näkymässä on esitelty vain päivässä suoritetut tunnit, viikon yhteenlaskettu tuntimäärä.

Uutta työtuntien näkymiskalenteria alettiin ensin toteuttaa löydetyn vapaan lähdekoodin JavaScript ja jQuery -kirjastojen avulla. Kuitenkin liittymä ei ollut tarpeeksi helppokäyttöinen ja lopulta päätettiin toteuttaa kalenteri PHP:lla, jolloin tietokannasta datan tuominen kävisi olemassa olevien luokkien käytön avulla paljon helpommin.

Poiketen aikaisemmasta ohjelmistosta, haluttiin työtuntien kalenterinäkymää uudistaa niin, että olisi mahdollista nähdä paljon enemmän tietoa tehdyistä työtunneista. Aiemman viikkonäkymän (kuva 28) mukaan ei selviä

juuri mitään tehdyistä töistä. Vasta klikkaamalla tuntien määrää, siirrytään päivänäkymään, jossa nähdään paremmin töiden erittely.



Kuva 29. Uuden ohjelmiston viikkonäkymä. Viikkonäkymässä kerrotaan koko viikolta tärkeää tietoa, suoritustaso, sekä näytetään eriytetysti kunkin päivän kohdalla tehdyt työtunnit. Lisäksi näkymien vaihtaminen on helppoa.

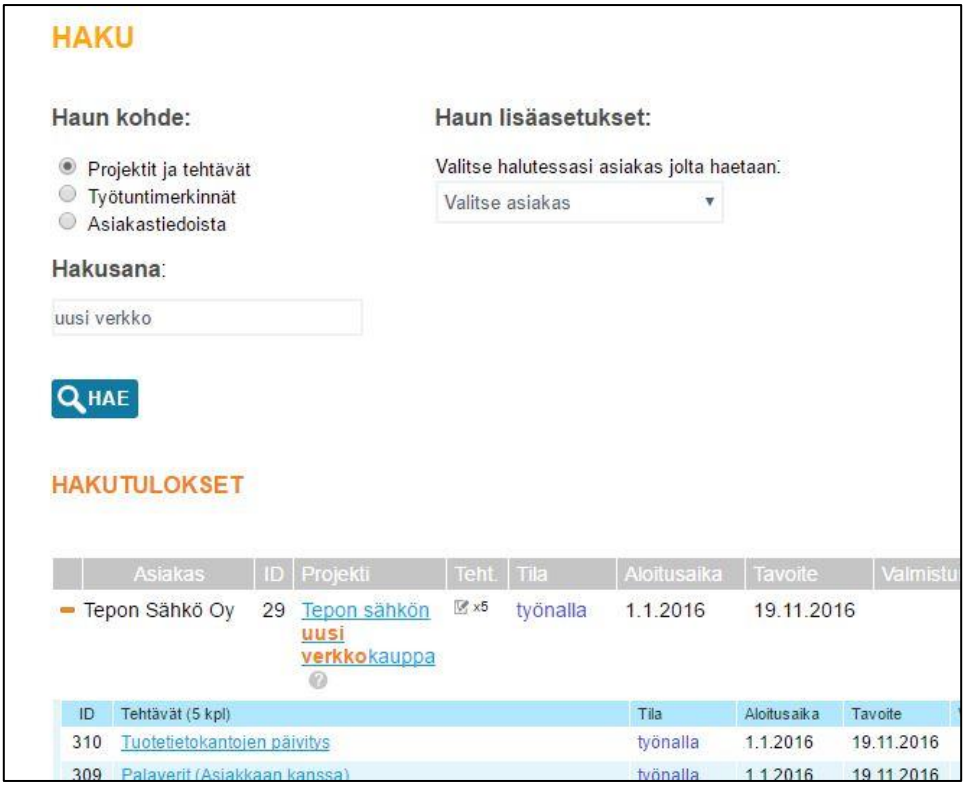
Uuteen ohjelmistoon suunnitellussa PHP -pohjaisessa kalenterinäkymässä on viikko ja päivänäkymän lisäksi myös kuukausinäkymä, joka helpottaa useamman viikon mittaisien projektien hahmottamista.

4.5 Seitsemäs iteraatio

Seitsemännessä iteraatiossa keskityttiin muihin toimintoihin mitkä ovat ohjelmiston toiminnan kannalta tärkeitä. Iteraation kesto oli noin neljätoista päivää.

4.5.1 Haku

Haun käyttäminen oli vanhan ohjelmiston yksi hitaimmista ja raskaimmista toiminnoista (silti yksi käytetyimmistä), sillä se kävi läpi valtavan joukon tietueita yhdeksän vuoden ajalta ja tulosti ne erittäin tehottomasti käyttäjälle.



Kuva 30. Hakutoiminto ohjelmiston betaversiossa

Haku haluttiin pitää mahdollisimman yksinkertaisena, sekä hyvin selkeästi ryhmiteltynä. Haun arvo muuttuu vanhasta ohjelmistosta selkeästi, sillä projektien ja tehtävien suodatuspaneelilla pyrittiin uudessa ohjelmistossa selvästi korvaamaan osa haun käytöstä. Ideana oli että käyttäjät käyttäisivät enemmän suodatuspaneelia kuin etsisivät projekteja tai tehtäviä pelkästään haun avulla. Suodatuspaneeli mahdollistaisi käyttäjälle nopean mahdollisuuden valita kaikki ne projektit näytettäväksi heti ohjelmistoon kirjautumisen jälkeen.

Hakuun lisättiin projektien ja tehtävien hakumahdollisuuden lisäksi mahdollisuus hakea suoraan asiakkaiden tiedoista tiettyä sanaa. Lisäksi avattiin mahdollisuus myös hakea työtuntimerkinnöistä, jolloin olisi esimerkiksi mahdollista etsiä omaa nimeään muiden työtunti-merkinnöistä.

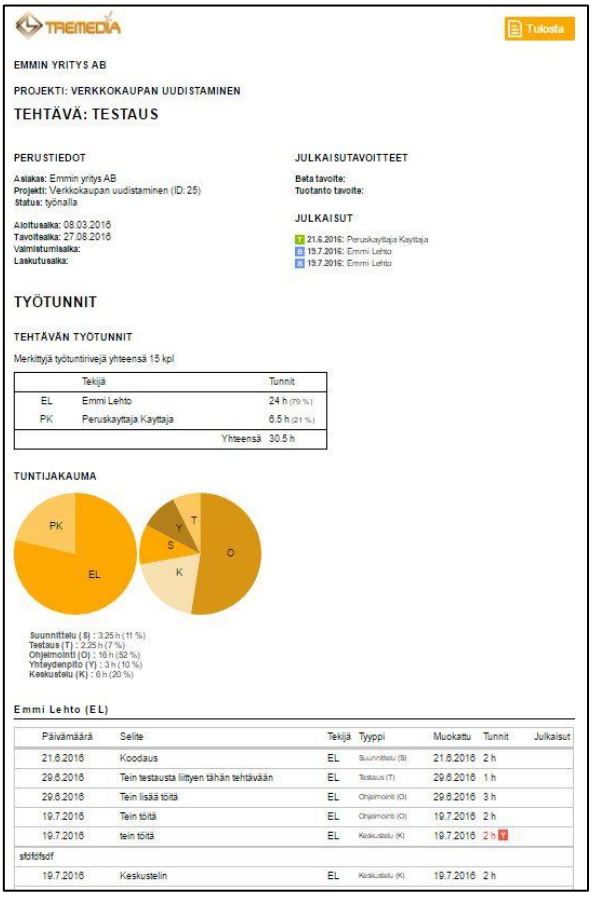
Esimerkiksi henkilö A on osallistunut keskusteluun henkilön C kanssa, mutta on unohtanut merkitä työtunnit keskustelusta. Henkilö C sen sijaan on merkinnyt työtunnit keskustelusta ja lisäksi merkinnyt että henkilö A on osallistunut keskusteluun kommenttirivissä. Tällöin henkilö A löytää kyseisen työtuntimerkinnän ohjelmiston hakutoiminnon kautta hakemalla omalla nimellään ja täten pystyy helpommin merkitsemään työtunteja myös niihin tehtäviin joissa hän on selkeästi ollut osallisena.

4.5.2 Raportit

Raportteja oli tarkoitus tehdä useampia uuteen ohjelmistoon, mutta koska aikataulullisesti usean raportin koostaminen oli hankalaa, päätettiin kesän

lopun kehityspalaverissa, että raportteja tehdään vain muutama ja niiden kehitys jatkuu sekä loppuvuodesta tai seuraavan vuoden alusta.

Tulostettavia raportteja tehtiin yhteensä 2 kappaletta betaversioon. Toinen on projektiraportti joka sisältää projektista oleellisia tietoja tulostettavassa muodossa. Toinen raportti on tehtäväraportti, joka sisältää tehtävän sekä siihen tehtyjen työtuntien tiedot.



Kuva 31. Tehtäväraportin ulkoasu on pelkistetty versio tehtävän hallinnan näkymästä.

Raporteista on pyritty karsimaan pois hallinnan värit, jotta tulosteet olisivat selkeitä myös mustavalkoisina. Lisäksi tulosteissa on pyritty yksinkertaisempaan esitysmuotoon, jolloin raporttien tulkitseminen on helpompaa.

4.6 Kahdeksas iteraatio

Kahdeksannen iteraation tarkoituksena oli optimoida, tarkistaa, testata ja korjata ohjelmistoon tehtyjä ominaisuuksia ja valmistella sitä käyttäjien betatestausta varten. Iteraation kesto oli noin neljätoista päivää.

4.6.1 Korjaukset ja testaus

Testauksessa edettiin aikaisempien iteraatioiden mukaan ja varmistettiin toimintojen laatu. Tähän kuului esimerkiksi virhe- ja muiden syötteiden tarkistaminen. Korjauksia tehtiin sen mukaan kuin testauksesta kirjatut ongelmat oli löydetty.

Varsinainen ohjelmiston testaaminen tapahtuisi betaversion julkistamisen jälkeen. Testaukseen tehtäisiin tuolloin erillinen suunnitelma. Testauksessa tarkoituksena olisi hyödyntää ainakin automatisoitua yksikkötestausta Selenium -ohjelmiston avulla ja tehdä lisäksi tietoturvatestausta Acunetix – ohjelmistolla. Kun ohjelmisto lopulta etenee testauksesta lopulliseen versioonsa, tapahtuu silloin ohjelmiston käyttöönotto.

4.6.2 Optimointi

Optimointiin kuului tietokantataulujen määrittelyn yhtenäistämistä ja korjauksia. Tavat muuttuivat rakennuksen aikana hieman, jolloin oli tärkeää käydä ensimmäiset taulut läpi ja varmistaa että tietueiden määrittelyt olisivat jokaisessa tietokannan taulussa samalla lailla.

Myös luokkia ja tiedostoja siistittiin yleisesti ja tämän lisäksi kommentoitiin ja eriteltiin ohjelmistokoodia. Erittelyllä tarkoitetaan tässä sitä, että PHP koodia otettiin tiedostosta ja se laitettiin paremmin sitä kuvaavaan PHP include - tiedostoon, jolloin eri moduulien löytäminen lähdekoodista olisi helpompaa kehittäjien näkökulmasta.

Optimoinnissa vertailtiin myös latausnopeuksia eri indikaattoreiden avulla ja sen perusteella tehtiin muutoksia sivun tietokantakyselyihin, sekä ohjelmistokoodin tarkistuksiin ja hakemistorakenteeseen. Optimoinnin tulokset olivat yllättävän näkyviä ja sen perusteella saatiin jopa kriittisempiäkin virheitä seulottua ohjelmistosta pois.

4.7 Ohjelmiston jatkokehitys

Koska ohjelmiston valmistumisen kannalta alkoi tulla aikataulullisesti kiire vuoden 2016 heinä-elokuussa, päätettiin että arkistoinnin keskeneräiset toiminnallisuudet tiputtaa pois betaversion julkaisusta. Arkistointitoiminnot kuitenkin siirtyivät jatkokehityslistalle ja ovatkin erittäin tärkeitä jotta varmistetaan ohjelmiston pysyvät eheänä vuodesta toiseen.

Suunnitelmissa oli toteuttaa ohjelmistoon paljon raportointiin liittyviä työkaluja, mutta betaversion aikana haluttiin antaa mahdollisuus jo tallentaa ohjelmistoon paljon sellaista tietoa, joista raporttien koostaminen myöhemmin olisi mahdollista. Käytännössä koska tietokantaan on jo tallennettu valmiiksi tieto esimerkiksi ylityötunneista, olisi mahdollisuus tehdä jatkokehityksenä esimerkiksi toiminto jolla voitaisiin tuoda asiakaskohtainen ylityötuntijakauma viimeisen vuoden ajalta. Tämän perusteella voitaisiin sitten projektinhallinnan kannalta analysoida, että minkä asiakkaan projekteihin on mahdollisesti varattu liian vähäisesti aikaa työn tekemiseen.

Betaversion testauksen kannalta kuitenkaan nämä jatkokehityslistalle siirtyneet toiminnot eivät olleet merkittäviä, koska niiden käyttäminen ja tarve tulisi vasta oleelliseksi kun ohjelmistossa olisi enemmän tietuetta analysoitavissa, eli siis enemmän projekteja, tehtäviä ja työtunteja.

5 YHTEENVETO JA JOHTOPÄÄTÖKSET

Ohjelmistojen uudistaminen voidaan nähdä erittäin tuottavana tapana saada nopealla aikavälillä laadukkaampia ohjelmistoja, jotka palvelevat käyttäjiä paremmin. Mielestäni uudistamisessa on tärkeää käyttää ketteriä kehitysmalleja, sillä ne auttavat paremmin ohjelmistoa suunniteltaessa päättämään esimerkiksi mitä komponentteja kannattaa hyödyntää nykyisestä järjestelmästä ja mitä komponentteja kannattaa mm. suunnitella tai ohjelmoida kokonaan riippumattomana nykyisestä ohjelmistosta.

Ohjelmiston toista versiota tehdessä näen tärkeänä myös aikaisemman version toimintomallien ja ominaisuuksien kyseenalaistamisen. Kun kyseenalaistamista tehdään tarpeeksi jokaisessa iteraatiossa, pystytään vanhaa versiota kehittämään paljon paremmin. Palaute ohjelmiston käyttäjiltä ja tilaajalta ohjaa jatkuvasti kutakin iteraatioita oikeaan suuntaan, ja tällöin virheet löydetään aikaisessa vaiheessa. Takaisinmallintaminen tukee vanhan ohjelmiston muuttamisen analysointia ja arkkitehtuurin arviointi antaa hyvän rungon uudelle ohjelmistolle.

Tärkeää on keskittyä yhden osa-alueen uudistamisen sijasta, uudistamaan ohjelmistoa kokonaisuudessa, jossa sen jokaista komponenttia on mietitty paremmaksi. Yleisesti tämä voisi tarkoittaa sitä, että uudistamisessa keskityttäisiin ohjelmistoarkkitehtuurin parantamiseen. Sen sijaan, että keskityttäisiin yksittäisen komponentin valmiiksi kehittämiseen, on parempi kehittää ja testata iteroiden. Tällöin alkuperäisessä suunnittelussa syntyneet virheet tai huonot käytännöt selviävät tarkennusvaiheessa.

Ohjelmistojen uudistamisessa tulisi ottaa huomioon myös ohjelman tulevat versiot ja uudelleenkäyttö. Kun ohjelmistojen versioissa aina pyritään siihen, että muutokset ovat myös mahdollisesti tehty niin että ne ovat parametri-kehitteistä tai moduulipainotteisia, voidaan ohjelmistoa tuotteistaa pienemmällä vaivalla myös tulevia ohjelmistoprojekteja varten.

Ohjelmistojen uudistamiseen liittyy paljon erilaisia huomioon otettavia asioita, jotka vaikuttavat siihen että millaiseen lopputulokseen halutaan päätyä. Uudistamisen tarpeen punnitseminen ja sen heijastaminen asiakkaan tai oman yrityksen liiketoiminnallisiin tavoitteisiin on tärkeää. Uudistamishankkeeseen ei kannata ryhtyä mikäli sillä ei todellisuudessa saavuteta liiketoiminnallista etua. Lisäksi ohjelmistojen uudistaminen tulisi aina miettiä osana yrityksen kokonaisvaltaista hanketta, jossa pyritään esimerkiksi tehostamaan yrityksen käytäntöjä, tai kuten tässä opinnäytetyössä, tehostamaan projektinhallinnan käytäntöjä pidemmällä aikajänteellä.

Uudistaminen voi sopia myös ohjelmistoon, josta yritys haaveilee saavansa liiketoiminnallisesti kannattavan tuoterungon markkinoille. Esimerkiksi Tremedian tapauksessa uutta ohjelmistoa pystyttäisiin jatkojalostamaan ja tuotteistamaan myös vaadittaessa jonkin pienen yrityksen käyttöön, ja näin jatkossa olisi mahdollista saada myös liikevoittoa ohjelmiston kehittämisestä. Kun kehittäminen tapahtuu saman tuoterungon päälle, pystyttäisiin sitä hyödyntämään myös yrityksen sisäisen ohjelmiston kehitykseen niin, että asiakasta voidaan laskuttaa samalla tehdystä työstä.

Ohjelmistojen uudistamisesta hankalaa tekee sen monimuotoisuus. Tekniikoita ja taktiikoita on paljon. Projektin aloitus voi olla haasteellista, varsinkin jos sitä jäädään liian pitkäksi aikaan määrittelemään. Määrittelyvaiheen pitkittyneisyys näkyy ohjelmiston kustannuksissa, jolloin nopeammin käyntiin polkaistu ketterien kehitysmenetelmien malli sopii paremmin niin pieniin kuin isoihin projekteihin. Iteratiivisuus ja inkrementaalisuus mahdollistavat samanaikaisen suunnittelun ja toteutuksen. Tämän etuna on se, että takaisinmallintamista ja vanhan ohjelmiston analyysiä tehdään samanaikaisesti. Tällä tavalla vanhasta ohjelmistosta pystytään mielestäni tehokkaammin analysoimaan tärkeimmät toiminnallisuudet ja siinä arvostetut ominaisuudet, kun taas häiritsevät piirteet pystytään samalla jättämään uudesta ohjelmistosta pois.

Kun uudistamista pohjustetaan esitutkimuksella, voi lopputulos olla jopa hyvinkin erilainen kuin alun perin on suunniteltu. Voi olla, että kokonaisvaltaisen uudistamisprojektin sijasta ohjelmistoa on mahdollista vain eheyttää, optimoida tai refaktoroida tehokkaasti, jolloin esimerkiksi uuden ohjelmiston tuottaminen on tarpeetonta. Toisaalta samalla punnistaan ohjelmiston käyttöarvo, eli esimerkiksi onko ohjelmisto tarpeeton yritykselle tai sisältääkö se tarpeettomia ominaisuuksia.

Uudistamiseen liittyy paljon riskejä ja arviointi on hankalaa. Projektin aikana huomattiin, että aikataulutukseen liittyi paljon haasteita liittyen siihen, että kehittäjä oli sidonnainen useampiin eri projekteihin. Kun eri iteraatioita tehtiin, tarkentui myös aikatauluarvio kesän mittaan realistisemmaksi. Kun aikataulun ongelmakohtiin pystyttiin reagoimaan hyvän projektiseurannan perusteella, projekti pysyi hyvin hallinnassa. Työtuntiarvio ylitettiin vain noin viidellä prosentilla projektissa. Viiden prosentin ylitystä voidaan pitää hyvänä saavutuksena puolen vuoden mittaisessa ohjelmistouudistamisprojektissa, jossa oli vain yksi sitoutunut kehittäjä.

Opinnäytetyö tarjosi tekijälle mahdollisuuden päästä käytännössä toteuttamaan omaa ohjelmistoprojektia. Lisäksi tärkeässä roolissa oli projektinhallintaan liittyvät ominaisuudet ja niiden analysointi varsinkin ohjelmistokehittäjän näkökulmasta.

LÄHTEET

Fong, J. S. 2015. Information Systems Reengineering, Intregration and Normalization. Springer. Viitattu 16.7.2016.

Fowler, M.;Beck, K.;& Brant, J. 2002. Refactoring: improving the design of existing code. Boston, MA, USA: Addison-Wesley Longman Publishing Co. Viitattu 10.9.2016.

Grubb, P.;& Armstrong, A. T. 2003. Software Maintenance: Consepts and Practise. World Scientific Publishing Co. Pte. Ltd. Viitattu 4.6.2016.

Haikala, I.;& Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Hämeenlinna: Talentum. Viitattu 7.8.2016.

Harsu, M. 2003. Ohjelmien ylläpito ja uudistaminen. Jyväskylä: Talentum Media Oy. Viitattu 1.10.2016.

Iivonen, J. 10.4.2008. Sampo Pankin korteilla ei pystynyt maksamaan eikä nostamaan rahaa. Helsingin Sanomat. Viitattu 6.8.2016.

Itkonen, J. 17.11.2003. Uudelleenkäyttö ja komponentit. Viitattu 6.8.2016.
http://www.mit.jyu.fi/opetus/kurssit/jot/2003/ukk/uudis.pdf

Järvenpää, T.;& Kankare, I. 2013. Veikö Moolok vallan? Talentum Media Oy. Viitattu 17.8.2016.

Kan, S. H. 2002. Metrics and Models in Software Quality Engineering. Viitattu 2.7.2016.

Kipponen, T. 2005. Ohjelmiston laadun parantaminen puutemittareiden avulla. Viitattu 3.9.2016.
ftp://ftp.cs.joensuu.fi/pub/Theses/2005_MSc_Kipponen_Teemu.pdf

Koskimies, K.;& Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Helsinki: Talentum. Viitattu 1.10.2016.

Laitinen, P. 25.3.2008. Sampo Pankki uudisti yllä järjestelmänsä. Helsingin Sanomat. Viitattu 16.9.2016

Lappalainen, E. 4.7.2008. Kilpailijoiden mukaan Sampo Pankki on menettelyt lähes 40 000 asiakasta. Helsingin Sanomat. Viitattu 16.9.2016.

Lehtonen, T.; Tuomivaara, S.; Rantala, V.; Känsälä, M.; Mäkilä, T.; Jokela, T.; Isomäki, M. Sulautettujen järjestelmien ketterä käsikirja. Viitattu 20.9.2016.
http://trc.utu.fi/embedded/kasikirja

Microsoft. Reverse engineer an existing database into a database model. Viitattu 3.9.2016
https://support.office.com/en-us/article/Reverse-engineer-an-existing-database-into-a-database-model-fb034862-acfc-45bc-88b2-f33d1e1f8614

Saarinen, J. 20.4.2015. Helsingin Sanomat. VR jäi ohjelmistoyhtiön loukkuun: edessä pakko-ostos vanhalta kumppanilta. Viitattu 1.8.2016.
http://www.hs.fi/talous/a1429415783782

sqa.net. ISO 9126 Software Quality Characteristics. Viitattu 13.8.2016.
http://www.sqa.net/iso9126.html

Tian, J. 2005. Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement. John Wiley & Sons, Inc. Viitattu 13.8.2016.

Tremedia. 2016. Viitattu 18.5.2016.
http://www.tremedia.fi/

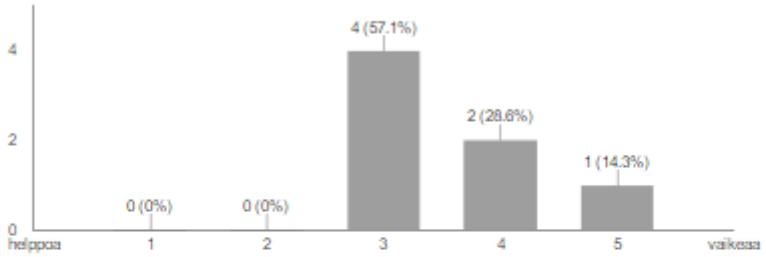
Tuovinen, A.-P. 2016. Ohjelmistoprosessit ja ohjelmistojen laatu. Helsingin yliopisto, Tietojenkäsittelytieteen laitos. Viitattu 4.7.2016.
https://www.cs.helsinki.fi/u/aptuovin/laatu/k16/Luento1.pdf

Vanhala, L. 30.1.2012. Näin VR sotki lippujärjestelmänsä – Miksi it-projektit epäonnistuvat? Suomen Kuvalehti. Viitattu 9.9.2016.

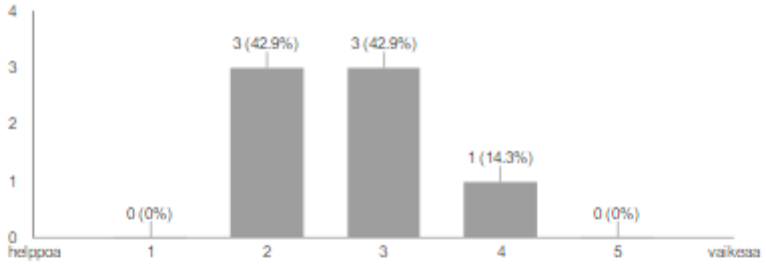
Yu, Y. 2005. Software Re-engineering. Viitattu 8.6.2016.
http://www.cs.toronto.edu/~yijun/ece450h/handouts/lecture2.pdf

Tremedia Intra – vanhan ohjelmiston arviointi, Liite 1
Vastaajat Tremedian työntekijät

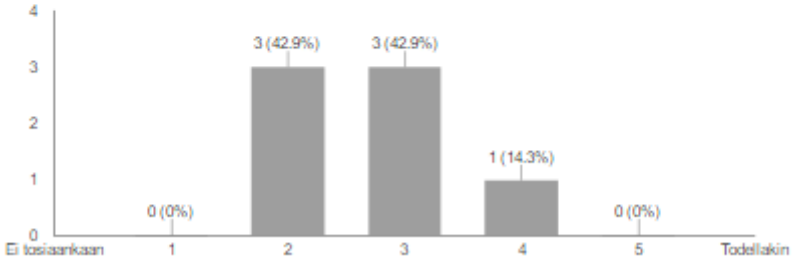
1. Työtehtävien / Projektien luominen ja määrittely on mielestäni (7 responses)



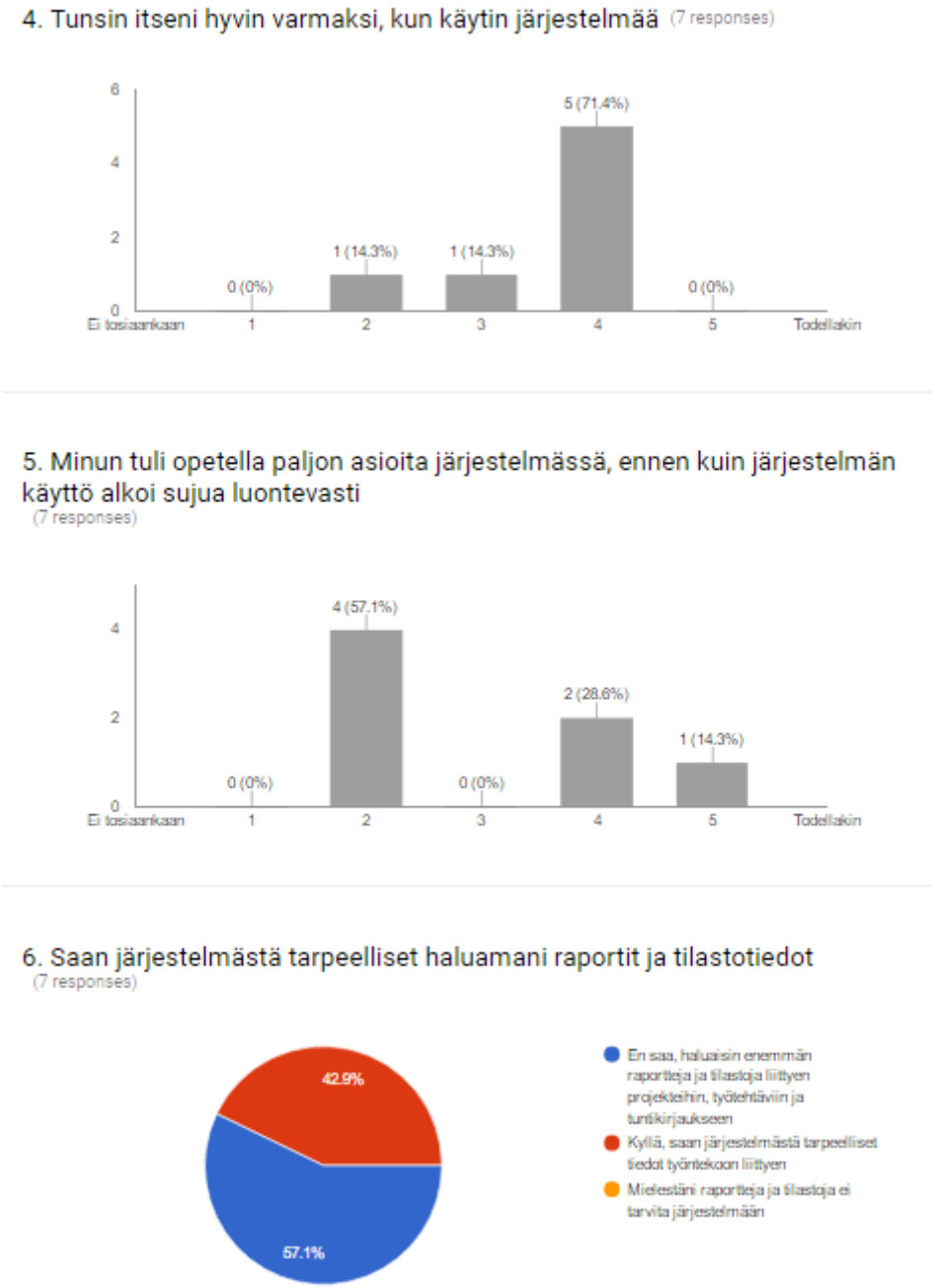
2. Työtuntien kirjaaminen on mielestäni (7 responses)



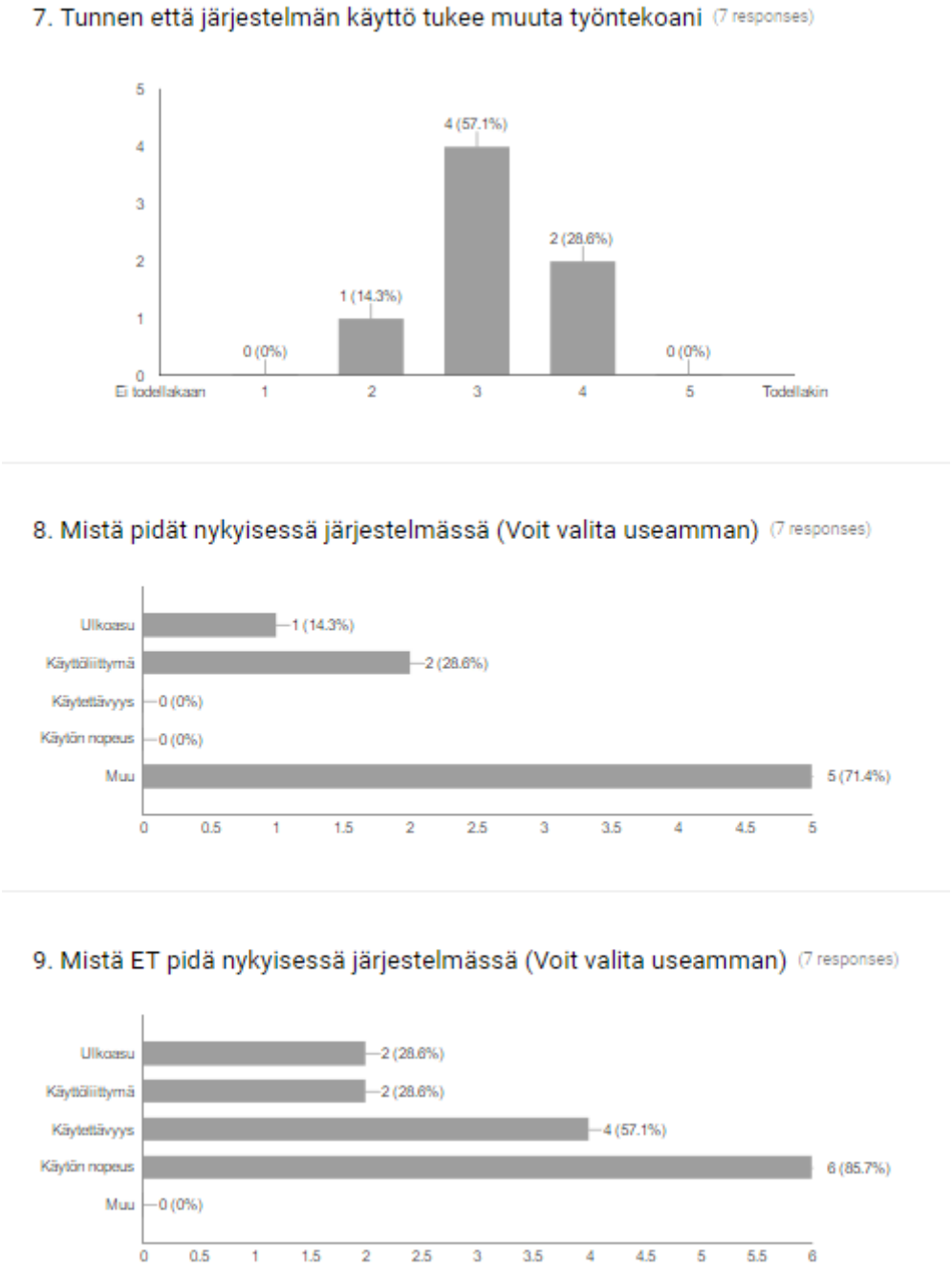
3. Järjestelmän käyttäminen on nopeaa ja löydän kaikki tarvittavat toiminnot helposti
(7 responses)



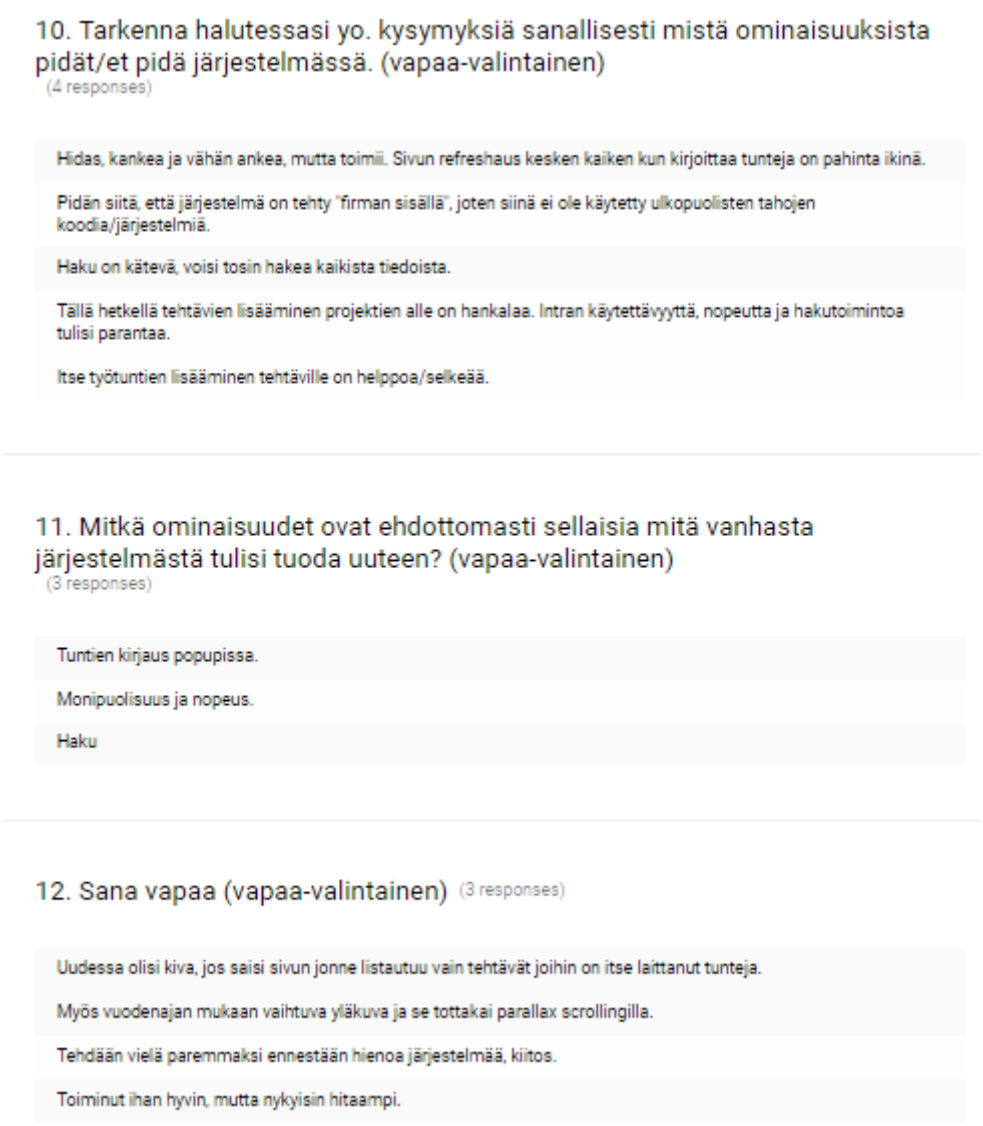
Kuva 32. Tremedia Intra – vanhan ohjelmiston arviointi 1/4



Kuva 33. Tremedia Intra – vanhan ohjelmiston arviointi 2/4



Kuva 34. Tremedia Intra – vanhan ohjelmiston arviointi 3/4



Kuva 35. Tremedia Intra – vanhan ohjelmiston arviointi 4/4